www.clubedelphi.net **Feita para desenvolvedores**

ClubeDelphi

Dando mais poder ao cliente
Como permitir que o próprio usuário da aplicação configure as fórmulas calculadas pelo sistema

Mão na Massa- Autenticação
Crie passo a passo uma aplicação que habilita/desabilita formulários dependendo do tipo de usuário que está logado

DevMedia GROUP Ano 6 · Edição 76 · R$8,90

ISSN 1517990-7

Mini-curso de Delphi e UML

Construa uma aplicação completa usando UML, POO e Persistência de Objetos

Auditoria
Faça um "check-up" no seu código-fonte

Padrões de Projeto e POO
Integração com o Together no Delphi 2006

Mini-Curso – Aplicação E-commerce Completa
Crie uma loja virtual com Delphi e ASP.NET – Parte III

CVS – Controle de versão
Conheça uma ferramenta free e integrada ao Windows Explorer

Especial- Biometria no Delphi

Crie passo a passo uma aplicação para reconhecimento de impressão digital

Proteja seu Software

ANTIPIRATARIA

DISTRIBUIÇÃO SEGURA

AUMENTO DOS SEUS LUCROS

PROTEÇÃO DA
PROPRIEDADE INTELECTUAL

Compact500®

Segurança. Qualidade. Eficiência.

Isto é a solução Compact-500. Você tem a garantia de proteção do seu software contra pirataria e engenharia reversa, além de controle total do gerenciamento de seu programa. Tudo isto de maneira fácil e extremamente eficiente.

Só uma empresa com um histórico de inovação, bom atendimento e sucesso pode garantir a segurança do seu bem mais valioso: suas informações.

Compact-500
A segurança do seu software

SafeNet®
The Foundation of Information Security

São Paulo (11) 3392 4600 www.safenet-inc.com/brasil

ClubeDelphi

Ano 6 - 76ª Edição - 2006 - ISSN 1517990-7

Impresso no Brasil

Editorial

A idéia da matéria de capa desta edição veio de uma experiência que tive há alguns anos, e também para preencher de uma vez por todas um vazio que existe entre POO e desenvolvimento RAD. Certa vez, vi um desenvolvedor Delphi criando um diagrama de classes. Esse diagrama possuía as entidades (classes), com seus respectivos atributos e operações (exemplo "Clientes"). Perguntei o que ele faria com o diagrama depois que começasse a desenvolver a aplicação no Delphi, e a resposta que tive foi a esperada. A menos que utilizemos um framework de persistência ou criemos nosso próprio mecanismo de mapeamento O/R (objeto/relacional), não há motivo para criar classes em um diagrama se formos tratá-las depois como tabelas. De fato, muitos desenvolvedores criam diagramas de "classes" quando poderiam (e deveriam) simplesmente criar um modelo ER simples. Normalmente não criamos uma classe chamada "TCliente" para acessar / mapear uma tabela chamada "Clientes" no banco de dados. O que fazemos é desfrutar do melhor do RAD oferecido pelo Delphi: colocamos uma classe pronta e genérica (o ClientDataSet claro) e acessamos qualquer tabela. E onde fica a POO? Qual a relação entre classe/tabela? Será que esse é o modelo correto de desenvolvimento com o Delphi? Poderia ser melhor?

Prezado leitor, confesso que há anos estava esperando por um mini-curso como esse, um casamento que até então parecia impossível: Modelagem UML, Programação Orientada a Objetos, Desenvolvimento RAD e Persistência, em uma aplicação totalmente real e prática! O resultado? Aplicações com mais qualidade, atendendo de forma mais efetiva aos requisitos, manutenção facilitada, melhor trabalho em equipe, menor impacto na mudança de requisitos, menor dependência de uma tecnologia e a certeza da satisfação do seu cliente com retorno do investimento.

Boa leitura e sucesso nos novos projetos OO.

Guinther Pauli – guinther@devmedia.com.br
Editor Geral

NÃO PERCA

PORTAL DO ASSINANTE

A ClubeDelphi tem uma novidade para você que comprou este exemplar na banca de jornal: você pode acessar GRATUITAMENTE, o Portal do Assinante ClubeDelphi!

Confira o que você encontra no Portal do Assinante:

- Mais de 130 Vídeo Aulas!
- Diversos Mini-Cursos Gratuitos!
- 1 Livro Eletrônico sobre ADO.NET e BDP!
- Mais de 130 Artigos Exclusivos!

Para Utilizar o Portal do Assinante, acesse www.devmedia.com.br/clubedelphi/portal.asp e utilize as informações abaixo:

Login: DVM.CD

Senha: X1LMZ

O acesso é válido por 30 dias a partida da data de lançamento da revista. Todos os meses a ClubeDelphi lhe dará uma senha válida para acessar o portal. Comprando a revista regularmente em bancas, você terá acesso ininterrupto a ele!

Corpo Editorial

Diretor Editorial e Administrativo
Gladstone Matos
gladstone@neoficio.com.br

Designer
Vinicius O. Andrade
viniciusoandrade@gmail.com

Capa
Antonio Xavier
jaxs_s_design@yahoo.com.br

Revisão
Fabio Correa
fabiocorrea@devmedia.com.br

Articulistas desta edição
Rafaela de Campos, Cristiano Caetano, Paulo Quicoli, Alberto G. S. Trevezani, Marco Antonio P. Araújo, Marcos Alexandre Miguel, Victory Fernandes, Everson Volaco

Na Web
www.devmedia.com.br/clubedelphi/portal.asp

Editor Geral
Guinther Pauli
guinther@devmedia.com.br

Editor Técnico
Luciano Pimenta
lucianopimenta@clubedelphi.net

Distribuição
Fernando Chinaglia Dist. S/A
Rua Teodoro da Silva, 907
Grajaú - RJ - 206563-900

Publicidade
Para informações sobre veiculação de anúncio na revista ou no site entre em contato com:

Kaline Dolabella
publicidade@devmedia.com.br

Fale com o Editor!

É muito importante para a equipe saber o que você está achando da revista: que tipo de artigo você gostaria de ler, que artigo você mais gostou e qual artigo você menos gostou. Fique a vontade para entrar em contato com os editores e dar a sua sugestão!
Se você estiver interessado em publicar um artigo na revista ou no site ClubeDelphi, entre em contato com os editores, informando o título e mini-resumo do tema que você gostaria de publicar:

Guinther Pauli - Editor da Revista
guinther@devmedia.com.br

Luciano Pimenta - Editor do Site
lucianopimenta@clubedelphi.net

Atendimento ao Leitor

A DevMedia conta com um departamento exclusivo para o atendimento ao leitor. Se você tiver algum problema no recebimento do seu exemplar ou precisar de algum esclarecimento sobre assinaturas, exemplares anteriores, endereço de bancas de jornal, entre outros, entre em contato com:

Thiago Andrade – Atendimento ao Leitor
www.devmedia.com.br/central/default.asp
(21) 2220-5375

Kaline Dolabella – Gerente de Marketing e Atendimento
kalined@terra.com.br
(21) 2220-5375

Parcerias

Para fechar parcerias ou ações específicas de marketing com a DevMedia, entre em contato com:

Jeff Wendell
jeff@devmedia.com.br

ASSESPRO PARANÁ

ClubeDelphi 76.indb 3 27.07.06 15:38:33

Ask The Expert

Perguntas e Respostas

Alterar a fonte de um painel do componente TStatusBar

Olá! Gostaria de saber como alterar a fonte de um painel, em um StatusBar do Windows.

Marcos Pereira

Postado no fórum da ClubeDelphi

Olá Marcos!

É muito fácil fazer o que você quer. O segredo está no evento *OnDrawTab* da classe *TCustomStatusBar*, que pode ser codificado para substituir o código padrão de desenho do componente. Veja no exemplo a seguir como fazer isso.

Inicie uma nova aplicação e coloque no formulário um *StatusBar*. Adicione a ele três painéis e defina a propriedade *Text* dos dois últimos para um texto qualquer. Ajuste também a propriedade *Width* para que seja possível visualizar o conteúdo dos painéis, deixando uma pequena "sobra" no valor definido.

Agora, troque a propriedade *Style* desses dois painéis para *psOwnerDraw*, para informar ao componente que a aplicação será responsável pelo seu desenho. Por fim, implemente o evento *OnDrawPanel* de *StatusBar* com o código a seguir:

```
procedure TMyForm.StatusBarDrawPanel(
  StatusBar: TStatusBar; Panel: TStatusPanel;
  const Rect: TRect);
begin
  with StatusBar do
  begin
  case Panel.Index of
    1:
      begin
        Canvas.Font.Style := [fsBold];
        Canvas.Font.Color := clRed;
      end;
    2:
      begin
        Canvas.Font.Style := [fsItalic];
        Canvas.Font.Color := clBlue;
      end;
  end;
  Canvas.FillRect(Rect);
  Canvas.TextOut(Rect.Left, Rect.Top, Panel.Text);
  end;
end;
```


ClubeDelphi 76.indb 4 27.07.06 15:38:34

Agora execute a aplicação. Veja na **Figura 1** como ficará o resultado visual final.

Tenha em mente que, por estarmos substituindo o manipulador padrão de desenho dos painéis, propriedades que afetam a aparência do componente não irão funcionar, como por exemplo, *Alignment*. É você que deve implementar o suporte a elas no evento *OnDrawPanel*.

Resposta enviada por **Michael Benford**

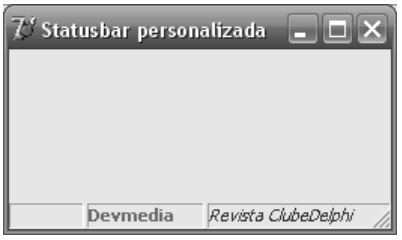


Figura 1. Barra de status personalizada

Campo TEXT no SQL Server e Delphi .NET

Gostaria de parabenizá-los pelo sensacional trabalho. Já estou conseguindo desenvolver alguma aplicação em .NET (BDS 2006). Naturalmente, estou tendo algumas dificuldades. Estou tentando utilizar um campo tipo TEXT (Memo) do SQL Server 2000 e não consigo mostrá-lo em nenhum Web Control (por exemplo, TextBox) a não ser com DBWeb Control, o que não gostaria, pois não estou utilizando DataSet.

Trata-se de uma simples consulta, onde utilizo um Command e um DataReader. Gostaria, se possível, que me orientassem quanto a visualização do campo TEXT em um componente Web Control + DataReader. Agradeço antecipadamente.

Richardson Leão Mendes
Via e-mail

Olá Richardson!

Use o seguinte código:

```
type
  TCharArray = Array of Char;
...
BdpConnection1.Open;
try
  dr := BdpCommand1.ExecuteReader;
  dr.Read;
  TextBox1.Text := Convert.ToString(TCharArray(dr['NOMECAMPO']));
finally
  BdpConnection1.Close;
end;
```

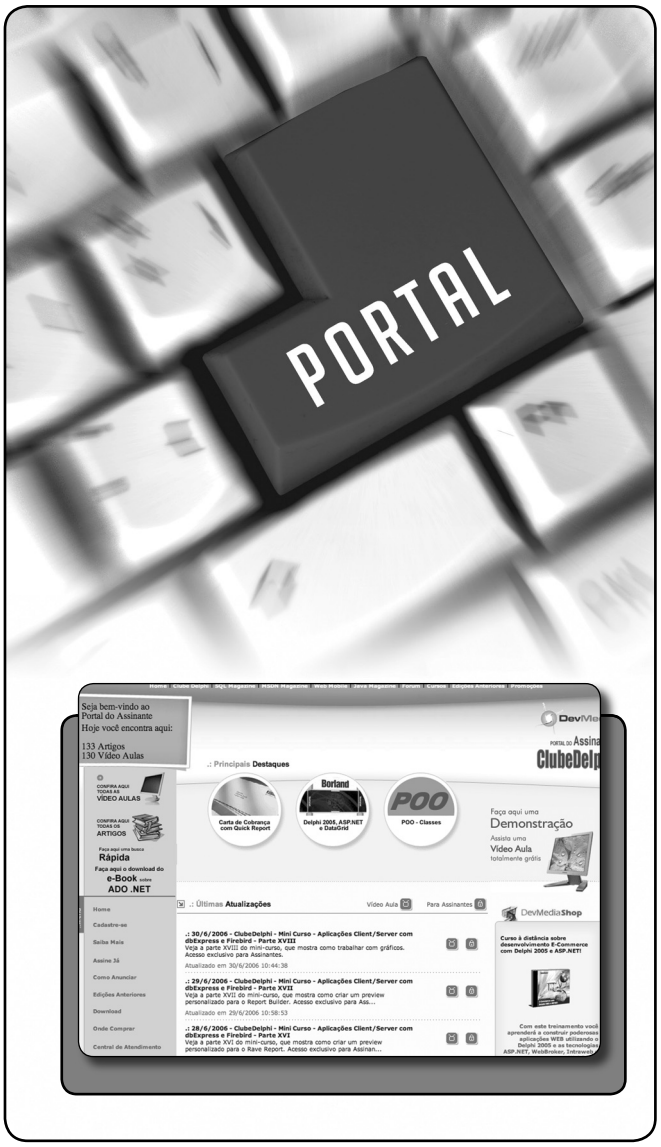
Caso não esteja usando o BDP, basta alterar o nome do provider que esteja utilizando.

Luciano Pimenta

ClubeDelphi 76.indb 5 27.07.06 15:38:34

Portal do Assinante

www.clubedelphi.net/portal



Caro Leitor,

O portal do assinante é a continuação, na Web, da revista ClubeDelphi. O portal recebe um conteúdo novo todo dia e hoje conta com: i) mais de 160 vídeo aulas; ii) diversos mini cursos gratuitos; iii) 1 livro eletrônico gratuito, de Guinther Pauli, sobre ADO.NET e BDP; iv) mais de 130 artigos exclusivos (que não foram publicados na revista)!;

Acesse o portal do assinante e receba muito mais conteúdo sobre Delphi! E o que é melhor: de graça! Todo leitor da revista ClubeDelphi, seja ele assinante ou comprador da revista em bancas, tem acesso ao portal do assinante (para quem compra em bancas, o acesso é válido por 30 dias).

Se você é assinante, utilize o seu login e senha pessoais para acessar o portal. Se você comprou em bancas, utilize o login e senha publicados na página do editorial desta edição.

Confira a seguir as últimas novidades do portal!

Boa leitura e sucesso!

Equipe DevMedia

Vídeo-Aulas

Mini Curso - Aplicações Client/Server com dbExpress e Firebird - Parte X

Veja a parte X do mini-curso, que mostra como trabalhar com consultas auxiliares em aplicações client/server.

Mini Curso - Aplicações Client/Server com dbExpress e Firebird - Parte XI

Veja a parte XI do mini-curso, que mostra como trabalhar com relatórios no Quick Report.

Mini Curso - Aplicações Client/Server com dbExpress e Firebird - Parte XII

Veja a parte XII do mini-curso, que mostra como trabalhar com relatórios no Rave Report.

ClubeDelphi 76.indb 6 27.07.06 15:38:35

Mini Curso - Aplicações Client/Server com dbExpress e Firebird - Parte XIII

Veja a parte XIII do mini-curso, que mostra como trabalhar com relatórios no Rave Report.

Mini Curso - Aplicações Client/Server com dbExpress e Firebird - Parte XIV

Veja a parte XIV do mini-curso, que mostra como trabalhar com relatórios no Report Builder.

Mini Curso - Aplicações Client/Server com dbExpress e Firebird - Parte XV

Veja a parte XV do mini-curso, que mostra como criar um preview personalizado para o Quick Report.

Mini Curso - Aplicações Client/Server com dbExpress e Firebird - Parte XVI

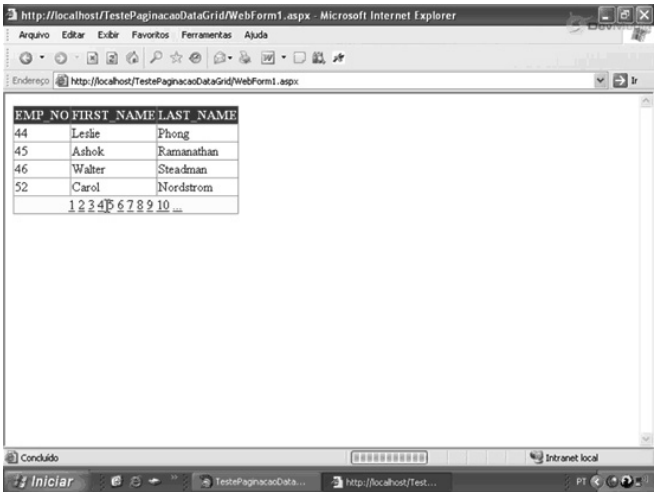
Veja a parte XVI do mini-curso, que mostra como criar um preview personalizado para o Rave Report.

Mini Curso - Aplicações Client/Server com dbExpress e Firebird - Parte XVII

Veja a parte XVII do mini-curso, que mostra como criar um preview personalizado para o Report Builder.

Mini Curso - Aplicações Client/Server com dbExpress e Firebird - Parte XVIII

Veja a parte XVIII do mini-curso, que mostra como trabalhar com gráficos.



Mini Curso - Aplicações Client/Server com dbExpress e Firebird - Parte XIX

Veja a parte XIX do mini-curso, que mostra como criar um login para validação de usuários.

Mini Curso - Aplicações Client/Server com dbExpress e Firebird - Parte XX

Veja a última parte do mini-curso, que mostra como fazer o deploy da aplicação usando o Inno Setup.

DataGrid Master/Detail no Delphi 2005

Veja nessa vídeo aula de Luciano Pimenta, como criar um Master/Detail com um único DataGrid.

Usando o Repeater no Delphi for .NET

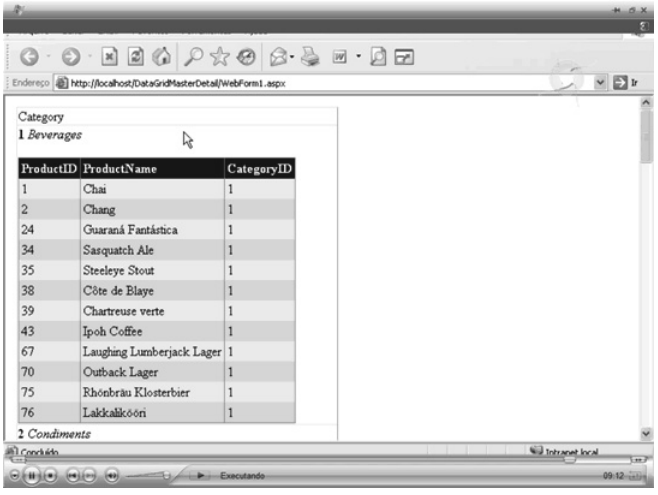
Veja nessa vídeo aula de Guinther Pauli, como utilizar o Repeater em aplicações ASP.NET.

Paginação do DataGrid

Veja nessa vídeo aula de Guinther Pauli, como criar paginação em um DataGrid em aplicações ASP.NET.

User Controls

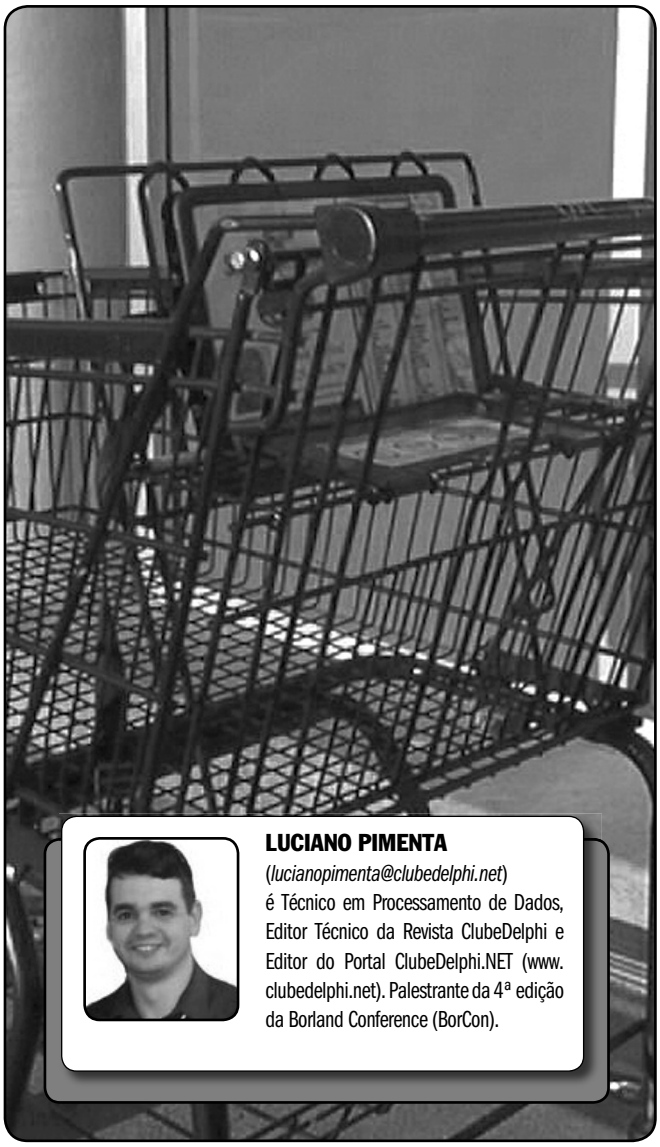
Veja nessa vídeo aula de Guinther Pauli, como trabalhar com User Controls no Delphi .NET.



ClubeDelphi 76.indb 7 27.07.06 15:38:38

Aplicação Web Completa

Carrinho de Compras - Parte 3



LUCIANO PIMENTA
(*lucianopimenta@clubedelphi.net*)
é Técnico em Processamento de Dados, Editor Técnico da Revista ClubeDelphi e Editor do Portal ClubeDelphi.NET (www.clubedelphi.net). Palestrante da 4ª edição da Borland Conference (BorCon).

Continuaremos neste artigo a criação da nossa aplicação Web completa, onde mostraremos como criar um carrinho de compras para o site. Para este artigo, veremos algumas funcionalidades de um dos componentes mais importante da arquitetura ADO.NET, o *DataSet*.

O *DataSet* trabalha totalmente desconectado do banco, com os dados em memória (cache), sendo de grande utilidade para criarmos o carrinho de compras, pois vamos adicionar os produtos comprados pelo cliente e no final incluiremos os mesmos no banco, sem a necessidade de a cada produto inserido, ser feita uma conexão com o banco.

Criando o carrinho de compras

Abra o projeto que estamos trabalhando, crie um novo *Web Form* (*File>New>Other>New ASP.NET Files>ASP.NET Page*) e dê o nome de "carrinho.aspx" para o mesmo. Adicione um *DataGrid* e configure-o para o estilo que mais lhe convier.

Adicione no formulário um *DataSet* (da categoria *Data Components*) e dê o nome de "dsCarrinho". Abra a propriedade *Tables*, no editor clique em *Add*, para adicionar uma "tabela" e em *TableName* digite "CARRINHO". Clique na propriedade *Columns* para adicionar as colunas da tabela, conforme a **Tabela 1.**

Nome da Coluna	Tipo (DataType)
ID_PRODUTO	System.Int32
NOME_PRODUTO	System.String
PRECO	System.Single
QUANTIDADE	System.Int32
SUBTOTAL	System.Single
TOTAL	System.Single

Tabela 1. Colunas da tabela CARRINHO do DataSet

ClubeDelphi 76.indb 8 27.07.06 15:38:39

Depois de criar as colunas, volte ao editor de *DataTables* do *DataSet* e acesse a propriedade *PrimaryKey*, marcando o campo ID_PRODUTO conforme visto na **Figura 1**.

Feche o editor. A configuração do *DataSet* está pronta, precisamos agora criar métodos e funções do nosso carrinho de compras.

Codificando o carrinho

Abra o editor e adicione o seguinte código na seção *private*:

```
procedure ConsultaProduto;
procedure AddProdutoCarrinho (IdProduto, NomeProduto,
  Preco, Quant: string);
procedure MostraGrid;
function GetDataSet: DataSet;
```

Aperte CRTL+SHIFT+C para que o Delphi crie o cabeçalho dos mesmos. A função *GetDataSet* simplesmente colocará nosso *DataSet* em sessão, com o seguinte código:

```
if (Session['CARRINHO'] = nil) then
  Session['CARRINHO'] := dsCarrinho;
Result := Session['CARRINHO'] as DataSet;
```

Se não colocássemos o *DataSet* em sessão, a cada nova requisição ele teria seus valores perdidos, ficando apenas o último registro adicionado. Este é um erro bastante comum nos desenvolvedores que estão iniciando a criação de aplicações ASP.NET com *DataSets*, lembre-se sempre de colocá-lo em sessão.

Repassaremos as informações do produto e quantidade através da URL da aplicação, com *QueryString*, como fizemos na edição anterior. O método responsável por selecionar as informações (dados) do produto é o *ConsultaProduto*.

Adicione um *FbConnection* no formulário e configure a conexão com o banco de dados. Adicione também um *FbCommand* ("cmdSelect"), faça a ligação ao *FbConnection* e adicione na propriedade *CommandText*, o seguinte código:

```
select ID_PRODUTO, NOME_PRODUTO, PRECO
from PRODUTOS
where ID_PRODUTO=?
```

Configure o parâmetro da consulta (dê o nome de "ID_PRODUTO") na propriedade *Parameters* do *cmdSelect*. No método *ConsultaProduto* adicione o código da **Listagem 1**.

Listagem 1. Método para consulta dos dados do Produto

```
var
  dr: FbDataReader;
begin
  FbConnection1.Open;
  try
    cmdSelect.Parameters[0].Value := Request.QueryString['CodProduto'];
    dr := cmdSelect.ExecuteReader;
    dr.Read;
    AddProdutoCarrinho(dr['ID_PRODUTO'].ToString,
      dr['NOME_PRODUTO'].ToString,
      dr['PRECO'].ToString,
      Request.QueryString['Quant'].ToString);
  finally
    FbConnection1.Close;
  end;
end;
```

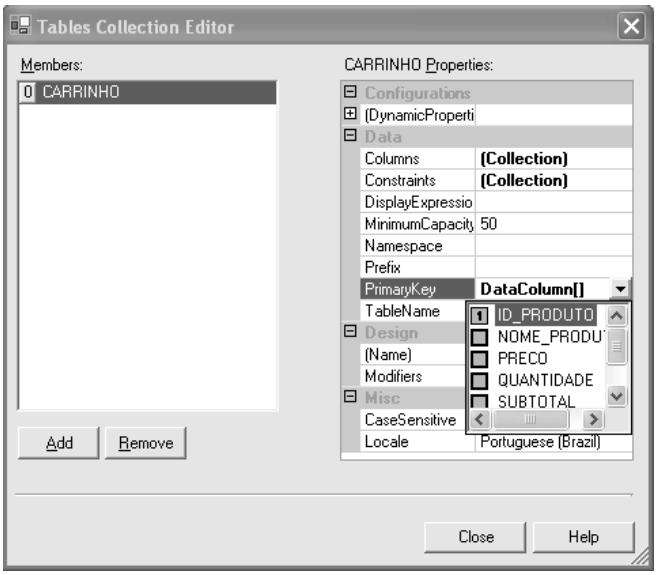


Figura 1. Escolhendo a chave primária da tabela

O código da listagem anterior faz a conexão com o banco, passa para o parâmetro da consulta o valor contido na *QueryString* e chama o *ExecuteReader*. Após, chamamos o método *AddCarrinho* passando os dados da consulta do produto. Esse método adiciona os valores dos parâmetros no *dsCarrinho*, conforme o código da **Listagem 2**.

Listagem 2. Método AddCarrinho

```
var
  dt: DataTable;
  row: DataRow;
begin
  dt := GetDataSet.Tables[0];
  row := dt.NewRow;
  row['ID_PRODUTO'] := IdProduto;
  row['NOME_PRODUTO'] := NomeProduto;
  row['PRECO'] := Preco;
  row['QUANTIDADE'] := Quant;
  dt.Rows.Add(row);
end;
```

O código adiciona uma nova linha no *dsCarrinho*. Note que não fizemos referência ao nome do *DataSet*, mas sim à função *GetDataSet* para que pegue o *DataSet* que está em sessão. No *MostraGrid* adicione o seguinte código:

```
DataGrid1.DataSource := GetDataSet;
DataGrid1.DataKeyField := 'ID_PRODUTO';
DataGrid1.DataBind;
```

Nesse código, apenas indicamos o *DataSource* do *DataGrid* e chamamos o *DataBind*. Vá até a página *visualizaproduto.aspx* e adicione um *TexBox* ("txtQuantidade) ao lado do botão *Adicionar ao Carrinho*. Coloque um *Label* para indicar que deve ser digitada a quantidade de produtos que deseja comprar.

Na propriedade *Text* do *txtQuantidade* configure "1", assim não precisamos validar se o usuário digitou um valor maior que zero (veja o box *Validações com JavaScript*).

ClubeDelphi 76.indb 9 27.07.06 15:38:41

Validações com JavaScript

Para validar que somente números sejam digitados em um TextBox, podemos usar os controles de validação vistos anteriormente neste curso. Aproveitarei para mostrar uma forma alternativa, a fim de demonstrar como usar JavaScript em aplicações ASP.NET. No *Load* da página *visualizaproduto.aspx* adicione o seguinte código:

```
txtQuantidade.Attributes.Add('onkeypress',
  'if (event.keyCode < 48 || event.keyCode > 57) '+
  '{event.keyCode = 0;}');
```

Com esse código o *txtQuantidade* aceitará somente números.

No evento *Click* do *ImageButton* altere o código já adicionado, para o seguinte valor:

```
Response.Redirect('carrinho.aspx?CodProduto='+
  Request.QueryString['CodProduto']+
  '&Quant='+txtQuantidade.Text);
```

Por fim, precisamos codificar o *Load* da página do carrinho para fazer a consulta do produto, adicionar o mesmo no *DataSet* e mostrar no *DataGrid*. Para isso, adicione o código da **Listagem 3** no *Page_Load*:

Listagem 3. Page_Load do carrinho.aspx

```
if not IsPostBack then
begin
  if Request.QueryString['CodProduto'] <> nil then
    ConsultaProduto;
  MostraGrid;
end;
```

Rode a aplicação e teste.

Melhorias na aplicação

Precisamos adicionar mais algumas validações/melhorias no nosso carrinho de compras, por exemplo, verificar antes de adicionar, se o produto não existe no *DataSet*, se existir, apenas precisamos alterar a quantidade do mesmo. Volte ao carrinho e vamos modificar o código do *AddCarrinho*, conforme a **Listagem 4.**

Listagem 4. Alterando o método AddCarrinho

```
var
...
  qtd: System.Int32;
begin
  dt := GetDataSet.Tables[0];
  dt.Rows.Find(IdProduto);
  if row = nil then
  begin
    row := dt.NewRow;
    row['ID_PRODUTO'] := IdProduto;
    row['NOME_PRODUTO'] := NomeProduto;
    row['PRECO'] := Preco;
    row['QUANTIDADE'] := Quant;
    dt.Rows.Add(row);
  end
  else
  begin
    qtd := row['QUANTIDADE'] as Int32;
    qtd := qtd + Convert.ToInt32(Quant);
    row['QUANTIDADE'] := System.&Object(qtd);
  end;
  dt.Columns['SUBTOTAL'].Expression := 'QUANTIDADE * PRECO';
  dt.Columns['TOTAL'].Expression := 'SUM(SUBTOTAL)';
end;
```

Veja que estamos verificando (*Find*) se o código do produto já existe. Caso seja verdadeiro, a variável (*row*) será diferente de *nil*, portanto, apenas alteramos o campo QUANTIDADE. Senão, adicionamos o registro no *DataSet*. No final, através da propriedade *Expression*, passamos o código para calcular os campos SUBTOTAL e TOTAL. Veja na **Figura 2** o carrinho de compras em execução.

Vamos adicionar o valor do campo TOTAL no rodapé do *DataGrid*, para isso, acesse o evento *ItemDataBound* do *DataGrid* e adicione o seguinte código:

```
if e.Item.ItemType = ListItemType.Footer then
  e.Item.Cells[3].Text := 'Total: '+
  GetDataSet.Tables[0].Rows[0]['TOTAL'].ToString;
```

Altere a propriedade *ShowFooter* do *DataGrid* para *True*. Adicione manualmente os campos no *DataGrid*, removendo os campos ID_PRODUTO e TOTAL. Faça também a configuração do campo PRECO para que mostre o valor no formato moeda (na propriedade *Data formatting expression*, digite "{0:c}"). Veja o resultado na **Figura 3**.

Para adicionar o formato de moeda no total do rodapé, basta alterar o *ItemDataBound* com o seguinte código:

```
e.Item.Cells[3].Text := 'Total: ' +
  System.&String.Format('{0:c}',
  GetDataSet.Tables[0].Rows[0] ['TOTAL']);
```

Figura 2. Carrinho de compras em uso

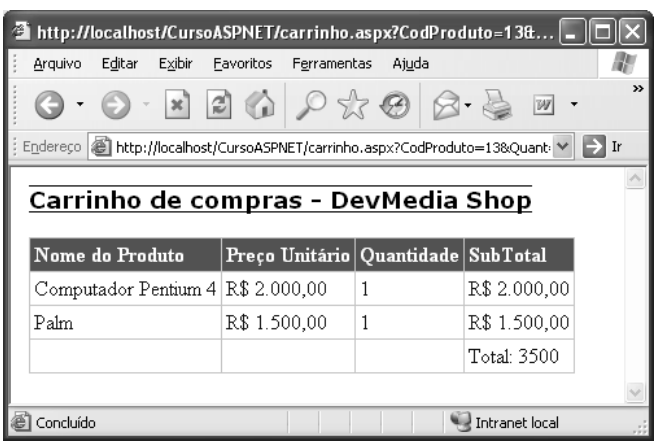


Figura 3. DataGrid mostrando o total do carrinho de compras no rodapé

ClubeDelphi 76.indb 10 27.07.06 15:38:42

Removendo produtos do carrinho

Vamos incluir a capacidade de remover um produto do carrinho. Abra o editor do *DataGrid* e adicione um botão do tipo *Button Column>Delete*. No evento *DeleteCommand* do *DataGrid*, adicione o código da **Listagem 5.**

Listagem 5. Removendo um produto do carrinho

```
var
  dt: DataTable;
  row: DataRow;
begin
  dt := GetDataSet.Tables[0];
  row := dt.Rows.Find(DataGrid1.DataKeys[
    e.Item.ItemIndex]);
  if row <> nil then
    dt.Rows.Remove(row);
  MostraGrid;
end;
```

A propriedade *DataKeys* do *DataGrid* armazena o valor do campo indicado na propriedade *DataKeyField*, que foi configurado no método *MostraGrid*. Assim, pesquisamos o valor do ID_PRODUTO e se for encontrado, removemos o mesmo do *DataSet*. Por fim, basta chamar novamente o *MostraGrid* para atualizar o carrinho.

Segurança

Nosso site de e-commerce não possui ainda qualquer validação para indicar qual usuário efetuou a compra. Vamos criar agora a tela de login da aplicação. Adicione um novo *Web Form* e dê o nome de "login.aspx". Adicione dois *TextBoxes* ("txtUser" e "txtSenha"), dois *Labels* e um botão. No *TextBox* de senha, altere a propriedade *TextMode* para *Password*. Veja na **Figura 4** como deve ficar o formulário.

Adicione também um *MessageBox*, um *FbConnection* e um *FbCommand*. Faça a ligação dos componentes de acesso a dados e utilize a seguinte instrução SQL no *FbCommand*:

```
select ID_USUARIO, SENHA
from USUARIOS
where UPPER(NOME_USUARIO) = ?
```

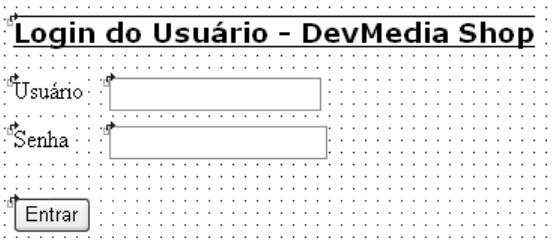


Figura 4. Criando a tela de login

Configure o parâmetro no *FbCommand* com o nome de "USUARIO". No botão de login, adicione o código da **Listagem 6.**

Listagem 6. Código para login da aplicação

```
var
  dr: FbDataReader;
begin
  FbConnection1.Open;
  try
    FbCommand1.Parameters[0].Value := txtUsuario.Text.ToUpper
    dr := FbCommand1.ExecuteReader;
    dr.Read;
    if (dr['SENHA'].ToString.Trim = txtSenha.Text) then
    begin
      Session['ID_USUARIO'] := dr['ID_USUARIO'];
      Response.Redirect(Session['URL'].ToString);
    end
    else
      MessageBox1.ShowMessage(
        'Usuário ou senha inválidos!');
  finally
    FbConnection1.Close;
  end;
end;
```

Note que se a senha conferir, salvamos em uma variável de sessão o código do usuário e redirecionamos para a página do carrinho, usando outra variável de sessão, que vai armazenar (veremos mais adiante na página *visualizaproduto.aspx*) a URL correta para inserção do produto no carrinho.

Normalmente, o usuário de um site e-commerce realiza o login ou faz um cadastro somente quando for finalizar a compra, ou no nosso exemplo, quando adicionar o primeiro item no carrinho. Dessa forma o usuário não precisa realizar o login para visualizar os produtos.

Mas, e se o usuário digitar diretamente no browser o caminho da página *carrinho.aspx*? Simplesmente, vamos verificar se o mesmo está logado, sabendo o valor da variável de sessão ID_USUARIO. No *Page_Load* do formulário do carrinho adicione o seguinte código:

```
if Session['ID_USUARIO'] = nil then
  Response.Redirect('login.aspx')
else
... <- código existente
```

Se ele não estiver logado, redirecionamos para a página de login. Essa técnica pode ser usada para qualquer página que não possa ser acessada por usuários anônimos.

Finalizando a compra

Para completar o nosso carrinho de compras, precisamos realizar algumas configurações. No User Control *header.ascx*, adicione no componente *Hyperlink* do carrinho, a página do carrinho de compras (*carrinho.aspx*), na proprie-

ClubeDelphi 76.indb 11 27.07.06 15:38:42

dade *NavigateURL*.

Se você tentar acessar a página por esse link, terá um erro, pois ainda não estamos verificando se queremos inserir um item no carrinho ou somente visualizar o mesmo. Para isso, altere o código do *Load* do *carrinho.aspx* com o código da **Listagem 7**.

Listagem 7. Alterando o código do Load da página carrinho.aspx

```
if Session['ID_USUARIO'] = nil then
  Response.Redirect('login.aspx')
else
begin
  if not IsPostBack then
  begin
    if Request.QueryString['Add'] <> nil then
    begin
      if Request.QueryString['CodProduto'] <> nil then
        ConsultaProduto;
    end;
    MostraGrid
  end;
end;
```

Note que estamos verificando mais uma vez a *Query String*. Na página *visualizaproduto.aspx*, também temos que modificar a URL, para que tenha a *QueryString* que indica que estamos adicionando um produto no carrinho, ficando com o seguinte código:

```
Session['URL'] := 'carrinho.aspx?Add=True&CodProduto='+
  Request.QueryString['CodProduto']+
  '&Quant='+txtQuantidade.Text;
Response.Redirect(Session['URL'].ToString);
```

Veja que nesse momento estamos salvando a URL para o carrinho. Quando for o primeiro item a ser adicionado, o usuário terá que logar, então a página *login.aspx* não sabe qual o produto escolhido pelo cliente, por isso estamos salvando o caminho (URL). Como já fizemos na página de login, após o usuário digitar a senha, repassamos esse caminho para que o item seja adicionado no carrinho de compras.

Adicione dois botões na página *carrinho.aspx*, um para voltar à página principal (para o usuário continuar comprando) e outro para finalizar a compra (crie um novo *Web Form* e chame-o de "finaliza.aspx".). Adicione dois *FbCommands* na página, dando o nome de "cmdVendas" e "cmdItens". Configure-os para apontar para o *FbConnection*.

Vamos criar duas *Stored Procedures* para adicionar os dados da venda e itens no banco. Abra o IBExpert e adicione o código da **Listagem 8** para criar a *Stored Procedure* de inserção de Vendas e de Itens.

Listagem 8. SPs de inserção de Vendas e Itens

```
CREATE PROCEDURE VENDAS_INS (
    ID_USUARIO INTEGER,
    DATA_VENDA DATE,
    TOTAL_VENDA NUMERIC(9,2))
AS
BEGIN
  INSERT INTO VENDAS (
    ID_VENDA,
    ID_USUARIO,
    DATA_VENDA,
    TOTAL_VENDA)
  VALUES (
    GEN_ID(GEN_VENDAS_ID, 1),
    :ID_USUARIO,
    :DATA_VENDA,
    :TOTAL_VENDA);
END

CREATE PROCEDURE ITENS_VENDAS_INS (
  ID_VENDA INTEGER,
  ID_PRODUTO INTEGER,
  QUANTIDADE INTEGER,
  VALOR_UNITARIO NUMERIC(9,2))
AS
BEGIN
  INSERT INTO ITENS_VENDAS (
    ID_VENDA,
    ID_PRODUTO,
    QUANTIDADE,
    VALOR_UNITARIO)
  VALUES (
    :ID_VENDA,
    :ID_PRODUTO,
    :QUANTIDADE,
    :VALOR_UNITARIO);
END
```

Voltando para o Delphi, configure a propriedade *Command Text* do *cmdVendas* para:

```
execute procedure VENDAS_INS (?, ?,?)
```

Crie os parâmetros com o mesmo nome dos campos (conforme fizemos em exemplos anteriores). Para o *cmdItens*, adicione o seguinte código na propriedade *CommandText*:

```
execute procedure ITENS_VENDAS_INS(?, ?, ?, ?)
```

Configure os parâmetros do componente. Adicione mais um *FbCommand* (chamado "cmdGenerator") para que possamos saber o valor do ID_VENDA, criado através de um *Generator* (configuração realizada no banco). Adicione o seguinte código no *CommandText* do *cmdGenerator*:

```
select GEN_ID(GEN_VENDAS_ID, 0) from RDB$DATABASE
```

Esse código pega o valor atual do *Generator*, passado como parâmetro. No botão de finalizar, vamos adicionar o código da **Listagem 9**.

Listagem 9. Finalizando as compras

```
var
  row: DataRow;
  IdCompra: System.&Object;
begin
  FbConnection1.Open;
  try
    cmdVenda.Parameters[0].Value := Session['ID_USUARIO'];
    cmdVenda.Parameters[1].Value := DateTime.Now;
      cmdVenda.Parameters[2].Value :=
        GetDataSet.Tables[0].Rows[0]['TOTAL'];
    cmdVenda.ExecuteNonQuery;
    IdCompra := cmdGenerator.ExecuteScalar;
    for row in GetDataSet.Tables[0].Rows do
    begin
      cmdItens.Parameters[0].Value := IdCompra;
      cmdItens.Parameters[1].Value := row['ID_PRODUTO'];
      cmdItens.Parameters[2].Value := row['QUANTIDADE'];
      cmdItens.Parameters[3].Value := row['PRECO'];
      cmdItens.ExecuteNonQuery;
    end;
  finally
    FbConnection1.Close;
  end;
  Session['CARRINHO'] := nil;
  Response.Redirect('finaliza.aspx');
end;
```

Na listagem anterior executamos a SP de Vendas e pegamos o valor do *Generator* (note que usamos o *ExecuteScalar*, pois teremos o retorno de somente uma linha e uma coluna, ficando assim mais otimizado).

Usamos o novo comando *for in* para varrer todas as colunas do *DataSet* e adicionar os valores dos itens da venda no respectivo *Command*. Por fim, fechamos a conexão com o banco, limpamos a sessão com o carrinho de compras (pois não será mais necessária) e redirecionamos para a página de finalização de compras. Para finalizar, você pode adicionar nas páginas os User Controls, criados anteriormente. Rode a aplicação e teste (**Figura 5**).

Conclusão

Vimos mais um capítulo do nosso mini-curso de ASP.NET e Delphi 2006. Aprendemos a criar o carrinho de compras, usando *DataSet* em memória, onde inserimos e excluímos os itens do carrinho. Adicionamos segurança ao site, pois o usuário pode visualizar os produtos do carrinho e conseqüentemente finalizar a compra, somente se estiver logado. Na próxima edição, vamos adicionar as informações referente a forma de pagamento, onde usaremos o CobreBem para gerar boletos na aplicação. Um grande abraço a todos e até a próxima!

■

Links

Curso de E-Commerce com Delphi 2005 e ASP.NET

www.devmedia.com.br/curso/ecommerce2005

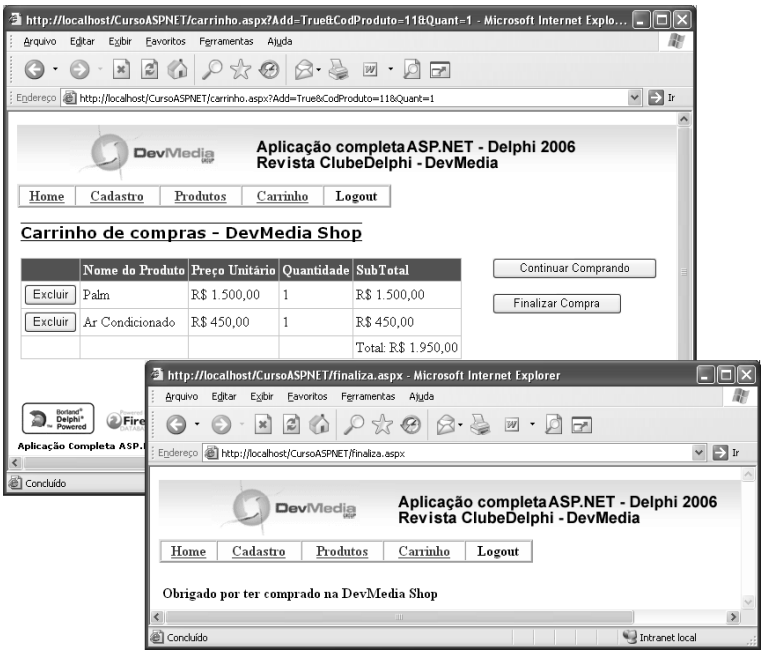
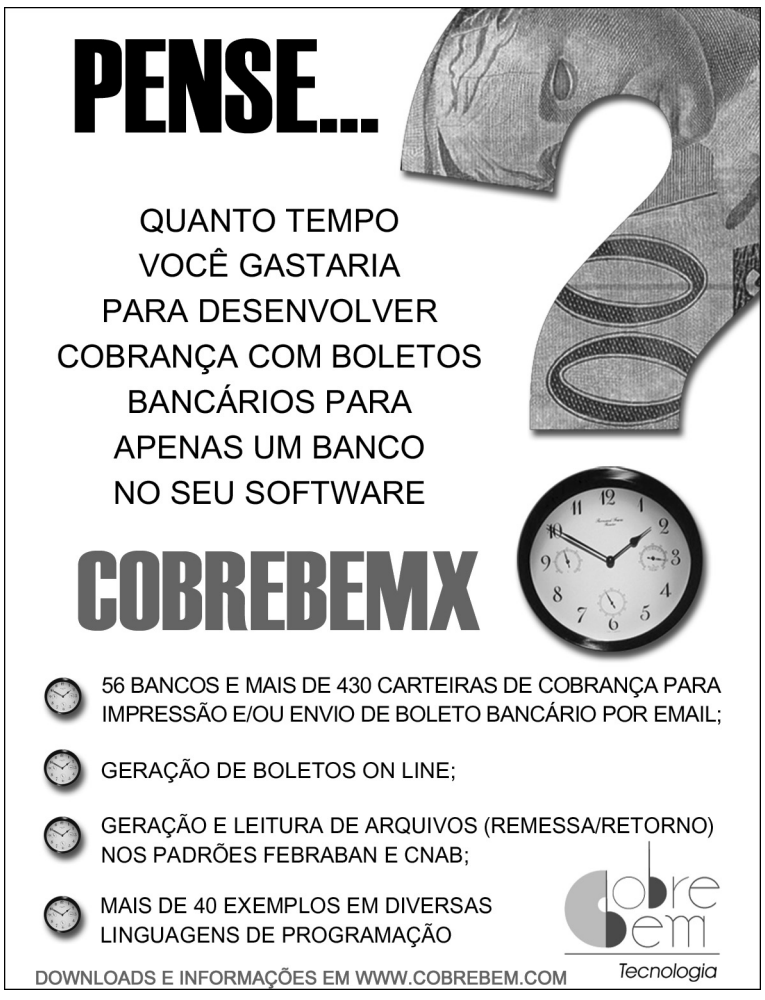


Figura 5. Finalizando a compra do site

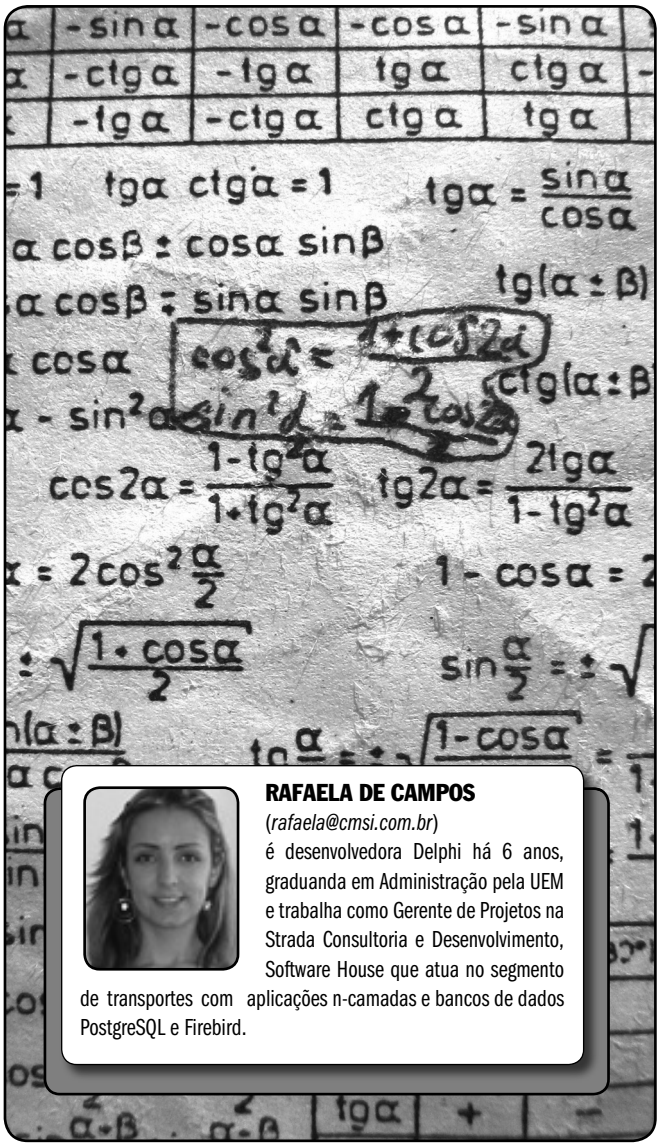


ClubeDelphi 76.indb 13 27.07.06 15:38:46

RAFAELA DE CAMPOS

Implementando fórmulas na aplicação

Agregando valor ao seu sistema através da personalização de cálculos internos



RAFAELA DE CAMPOS
(*rafaela@cmsi.com.br*)
é desenvolvedora Delphi há 6 anos, graduanda em Administração pela UEM e trabalha como Gerente de Projetos na Strada Consultoria e Desenvolvimento, Software House que atua no segmento de transportes com aplicações n-camadas e bancos de dados PostgreSQL e Firebird.

Customização é uma das palavras da vez e a cada dia mais o mercado exige que os produtos além de ter qualidade, sejam flexíveis, personalizados e tenham preços competitivos. Agora, como fazer para oferecer um sistema de qualidade, com preço competitivo e que sirva para diversos clientes com regras de negócios diferentes?

A resposta é: parametrização. Uma das maneiras de parametrizar o sistema é possibilitar que os cálculos sejam definidos através de fórmulas criadas pelos próprios clientes. E para demonstrar como utilizar fórmulas dentro de um sistema, neste artigo criaremos uma aplicação simples de compra e venda onde os cálculos do valor total do movimento e do item serão obtidos através do resultado da fórmula parametrizada.

Será usado o *Evaluator* da biblioteca de componentes JEDI, que é um avaliador de expressões para realizar o cálculo das fórmulas do programa, ou seja, ela decodifica e substitui expressões por seus respectivos valores numéricos.

Além disso, teremos uma tela para cadastro da fórmula digitada herdada de um formulário básico de cadastro, demonstrando assim também como utilizar herança de formulários dentro do Delphi.

Obs.: O objetivo deste artigo é demonstrar como tornar uma aplicação flexível e personalizável, não se atendo a princípios básicos, como criar um banco de dados, fazer conexão com banco, criar formulários na aplicação ou relacionamentos Master/Detail.

A classe TEvaluator

A classe *TEvaluator* é definida na unit *JclExprEval* da biblioteca JEDI e é uma classe avaliadora de expressões. Ela, assim como as outras classes de avaliadores de expressão

ClubeDelphi 76.indb 14 27.07.06 15:38:47

da JCL, é descendente da classe *TEasyEvaluator*, também definida na mesma unit. A declaração das duas classes está descrita na **Listagem 1**.

Listagem 1. Declaração das classes TEasyEvaluator e TEvaluator

```
...
TEasyEvaluator = class(TObject)
  private
    FOwnContext: TExprHashContext;
    FExtContextSet: TExprSetContext;
    FInternalContextSet: TExprSetContext;
  protected
    property InternalContextSet: TExprSetContext
      read FInternalContextSet;
  public
    constructor Create;
    destructor Destroy; override;

    procedure AddVar(const AName: string;
      var AVar: TFloat32); overload;
    procedure AddVar(const AName: string;
      var AVar: TFloat64); overload;
    procedure AddVar(const AName: string;
      var AVar: TFloat80); overload;

    procedure AddConst(const AName: string;
      AConst: TFloat32); overload;
    procedure AddConst(const AName: string;
      AConst: TFloat64); overload;
    procedure AddConst(const AName: string;
      AConst: TFloat80); overload;

    procedure AddFunc(const AName: string;
      AFunc: TFloat32Func); overload;
    procedure AddFunc(const AName: string;
      AFunc: TFloat64Func); overload;
    procedure AddFunc(const AName: string;
      AFunc: TFloat80Func); overload;
    procedure AddFunc(const AName: string;
      AFunc: TUnary32Func); overload;
    procedure AddFunc(const AName: string;
      AFunc: TUnary64Func); overload;
    procedure AddFunc(const AName: string;
      AFunc: TUnary80Func); overload;
    procedure AddFunc(const AName: string;
      AFunc: TBinary32Func); overload;
    procedure AddFunc(const AName: string;
      AFunc: TBinary64Func); overload;
    procedure AddFunc(const AName: string;
      AFunc: TBinary80Func); overload;
    procedure AddFunc(const AName: string;
      AFunc: TTernary32Func); overload;
    procedure AddFunc(const AName: string;
      AFunc: TTernary64Func); overload;
    procedure AddFunc(const AName: string;
      AFunc: TTernary80Func); overload;
    procedure Remove(const AName: string);

    procedure Clear;
    property ExtContextSet: TExprSetContext
      read FExtContextSet;
  end;

  TEvaluator = class(TEasyEvaluator)
  private
    FLexer: TExprSimpleLexer;
    FParser: TExprEvalParser;
  public
    constructor Create;
    destructor Destroy; override;
    function Evaluate(const AExpr: string): TFloat;
  end;
...
```

Para que possamos entender o funcionamento da classe *TEvaluator* é importante estudarmos o significado de três conceitos dentro do contexto: Variáveis, Constantes e Funções.

- *Variáveis*: São entidades que representam valores e podem ter seu valor alterado;
- *Constantes*: Assim como as variáveis, representam valores, porém, não podem ser alteradas;
- *Funções*: Representam chamadas a funções internas da aplicação.

Com base nisso, veremos então os métodos principais da classe:

- *AddVar*: Adiciona ao interpretador de expressões uma variável cujo nome será passado como uma *string* e representará a variável interna passada como parâmetro;
- *AddConst*: Da mesma forma que a *AddVar*, passa uma constante para o interpretador, cujo nome será passado como primeiro parâmetro e a constante que representa, o segundo parâmetro;
- *AddFunc*: Adiciona ao interpretador chamada a funções do sistema, que poderão receber até 3 parâmetros, sendo:
  - *TFloatFunc* (sem parâmetros): function Exemplo: Double;
  - *TUnaryFunc* (1 parâmetro): function Exemplo(x: Double): Double;
  - *TBinaryFunc* (2 parâmetros): function Exemplo(x,y: Double): Double;
  - *TTernaryFunc* (3 parâmetros): function Exemplo (x,y,z: Double): Double.

E, por último, porém não menos importante, temos o método *Evaluate* responsável pela execução da fórmula propriamente dita.

Criando o Banco de Dados

No IBExpert, criaremos o banco, chamado "Formulas.fdb", com suas respectivas tabelas, constantes na **Tabela 1.**

Tabela Formulas		
Campo	**Tipo**	**Tamanho**
CODFRM	VarChar	15
DESCRICAO	VarChar	50
TEXTO	VarChar	255
Tabela Movimentos		
Campo	**Tipo**	**Tamanho**
IDMOV	Integer	–
NOMECLIFOR	VarChar	50
DTEMISSAO	Date	–
VLROPCIONAL1	Numeric	15,2
VLROPCIONAL2	Numeric	15,2
VLRTOTALMOV	Numeric	15,2
Tabela ItensMovimentos		
Campo	**Tipo**	**Tamanho**
IDMOV	Integer	–
IDITM	Integer	–
QTDE	Numeric	15,2
VLRUNIT	Numeric	15,2
VLROPCIONAL1	Numeric	15,2
VLROPCIONAL2	Numeric	15,2
VLRICMS	Numeric	15,2
VLRIPI	Numeric	15,2
VLRTOTALITEM	Numeric	15,2

Tabela 1. Tabelas do banco de dados do exemplo

ClubeDelphi 76.indb 15 27.07.06 15:38:49

Serão mencionados somente os campos principais de cada uma das tabelas e não nos preocuparemos com chaves estrangeiras ou normalização do banco de dados. Nas tabelas *Movimentos* e *ItensMovimentos* temos dois valores opcionais que podem representar descontos, acréscimos, fretes ou quaisquer outros valores envolvidos nos cálculos dos totais.

Na tabela *ItensMovimentos*, além dos valores opcionais temos também os valores de impostos incidentes no item, onde o valor do ICMS não fará parte do total do item e o valor do IPI somente entrará no valor da nota.

Criando a aplicação

No Delphi, vamos criar uma nova aplicação e salvá-la como "P_Exemplo.dpr". No formulário principal adicionaremos um *MainMenu* que chamará os demais formulários da aplicação (informe o *Caption* do itens como "Fórmulas" e "Movimentos").

Renomeie o formulário criado por padrão, como "frmPrincipal". A chamada dos itens do menu está descrita na **Listagem 2** (lembre-se que esses formulários serão criados mais adiante no artigo).

Listagem 2. Chamadas dos itens de menus do frmPrincipal

```
procedure TfrmPrincipal.Formulas1Click(
  Sender: TObject);
begin
  frmCadFormulas.Show;
end;

procedure TfrmPrincipal.Movimentos1Click(
  Sender: TObject);
begin
  frmCadMovimentos.Show;
end;
```

Criaremos também um Data Module chamado "dmDados" (salve-o como "udmDados.pas"), para a conexão ao banco usaremos componentes da paleta *dbExpress* conforme mostrado na **Figura 1**.

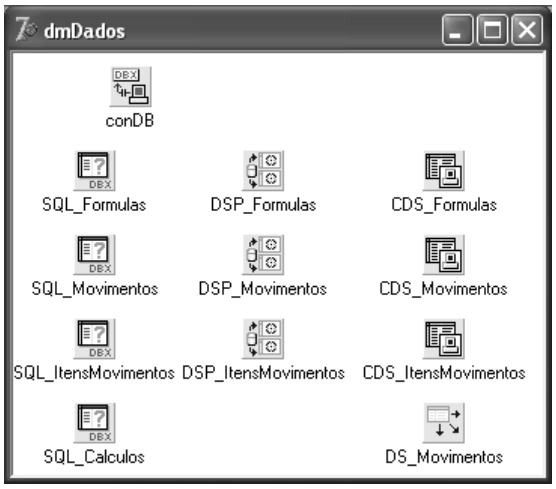


Figura 1. Componentes do dbExpress para o acesso ao banco

> **Nota:** Não entrarei em detalhes sobre a conexão dos componentes dbExpress. Para saber como criar uma conexão, veja edições anteriores da ClubeDelphi ou o mini-curso disponível no Portal do Assinantes no site da revista.

Na **Listagem 3** temos o código de métodos e eventos de componentes do *dmDados*.

No código da listagem anterior, temos o método *CalculaFormula* (que será explicado em maiores detalhes nos próximos tópicos), e a codificação dos eventos *BeforePost* e *AfterPost* da *CDS_Movimentos* e *CDS_ItensMovimentos*. No *BeforePost* temos a chamada à fórmula fazendo com que os campos de totais recebam o resultado dela, e no *AfterPost* faremos a chamada ao método *ApplyUpdates*, para que os dados sejam gravados no banco.

Podemos observar em nosso código que no evento *BeforePost* tanto da *CDS_ItensMovimentos* quanto do *CDS_Movimentos*, só ocorrem chamadas à função *CalculaFormula* e seu resultado é passado como valor para o campo de total da tabela.

A diferença entre os dois eventos é que no *BeforePost* da *CDS_Movimentos* temos que recalcular os totais de todos os itens antes de calcular o total do movimento, pois assim nos certificamos que os itens encontram-se com seus valores atualizados e que o total do movimento estará correto.

Continuando, a ligação entre as tabelas de *Movimentos* e *Itens_Movimentos* (para criar um Master/Detail) é mostrada na **Figura 2.**

O *SQL_Calculos* será usado para cálculo da soma dos totais dos itens de movimentos e sua propriedade *SQL* deverá ter o código da **Listagem 4.**

Figura 2. Ligação entre CDS_ItensMovimentos e CDS_Movimentos

ClubeDelphi 76.indb 16 27.07.06 15:38:50

Listagem 3. Código do dmDados

```
...
 public
    function CalculaFormula(CodFormula: string): Double;
  end;
...
uses U_FuncoesFormulas, JclExprEval, Dialogs;
...
function TdmDados.CalculaFormula(CodFormula: string): Double;
var
  { Variáveis do movimento }
  MvtVlrOpcional1, MvtVlrOpcional2,
  MvtVlrTotal: Double;
  { Variáveis do item de movimento }
  ItmVlrUnit, ItmQtde, ItmVlrTotal, ItmVlrICMS,
  ItmVlrIPI, ItmVlrOpcional1, ItmVlrOpcional2: Double;
  { Variáveis do cálculo }
  Evaluator: TEvaluator;
  Formula: string;
begin
  evaluator := TEvaluator.Create;
  try
    { Variáveis do movimento }
    evaluator.AddVar('MvtVlrOpcional1', MvtVlrOpcional1);
    evaluator.AddVar('MvtVlrOpcional2', MvtVlrOpcional2);
    evaluator.AddVar('MvtVlrTotal', MvtVlrTotal);
    { Funções do movimento }
    evaluator.AddFunc('SItmVlrTotal', SomaItmVlrTotal);
    evaluator.AddFunc('SItmVlrOpcional1', SomaItmVlrOpcional1);
    evaluator.AddFunc('SItmVlrOpcional2', SomaItmVlrOpcional2);
    evaluator.AddFunc('SItmVlrICMS', SomaItmVlrICMS);
    evaluator.AddFunc('SItmVlrIPI', SomaItmVlrIPI);
    { Variáveis do item de movimento }
    evaluator.AddVar('ItmVlrTotal', ItmVlrTotal);
    evaluator.AddVar('ItmVlrOpcional1', ItmVlrOpcional1);
    evaluator.AddVar('ItmVlrOpcional2', ItmVlrOpcional2);
    evaluator.AddVar('ItmVlrIPI', ItmVlrIPI);
    evaluator.AddVar('ItmVlrICMS', ItmVlrICMS);
    evaluator.AddVar('ItmVlrUnit', ItmVlrUnit);
    evaluator.AddVar('ItmQtde', ItmQtde);

    { Localizar na tabela de fórmulas o texto do
      código da fórmula passada como parâmetro }
    if not CDS_Formulas.Active then
      CDS_Formulas.Open;

    CDS_Formulas.Locate('CODFRM',CodFormula,[]);
    Formula := CDS_FormulasTEXTO.AsString;

    { Passagem de Valores para as variáveis }
    if pos('MvtVlrOpcional1', Formula) > 0 then
      MvtVlrOpcional1 := CDS_MovimentosVLROPCIONAL1.AsFloat;
    if pos('MvtVlrOpcional2', Formula) > 0 then
      MvtVlrOpcional2 := CDS_MovimentosVLROPCIONAL2.AsFloat;
    if pos('MvtVlrTotal', Formula) > 0 then
      MvtVlrTotal := CDS_MovimentosVLRTOTALMOV.AsFloat;

    if pos('ItmVlrTotal', Formula) > 0 then
      ItmVlrTotal := CDS_ItensMovimentosVLRTOTALITEM.AsFloat;
    if pos('ItmVlrOpcional1', Formula) > 0 then
      ItmVlrOpcional1 := CDS_ItensMovimentosVLROPCIONAL1.AsFloat;
    if pos('ItmVlrOpcional2', Formula) > 0 then
      ItmVlrOpcional2 := CDS_ItensMovimentosVLROPCIONAL2.AsFloat;
    if pos('ItmVlrIPI', Formula) > 0 then
      ItmVlrIPI := CDS_ItensMovimentosVLRIPI.AsFloat;
    if pos('ItmVlrICMS', Formula) > 0 then
      ItmVlrICMS := CDS_ItensMovimentosVLRICMS.AsFloat;
    if pos('ItmVlrUnit', Formula) > 0 then
      ItmVlrUnit :=  CDS_ItensMovimentosVLRUNIT.AsFloat;
    if pos('ItmQtde', Formula) > 0 then
      ItmQtde := CDS_ItensMovimentosQTDE.AsFloat;
    try
      Result := evaluator.Evaluate(Formula);
    except
      Showmessage('Erro ao executar fórmula. '+
        'Verifique a sintaxe digitada.');
      Result := 0;
    end;
  finally
    evaluator.Free;
  end;
end;

procedure TdmDados.CDS_ItensMovimentosBeforePost(
  DataSet: TDataSet);
begin
  CDS_ItensMovimentosVLRTOTALITEM.AsFloat := CalculaFormula('002');
end;

procedure TdmDados.CDS_MovimentosBeforePost(
  DataSet: TDataSet);
begin
  with CDS_ItensMovimentos do
  begin
    DisableControls;
    First;
    while not Eof do
    begin
      { Somente chamar o Edit e Post para que ele
        passe no BeforePost e refaça os cálculos }
      Edit;
      Post;
      Next;
    end;
    EnableControls;
  end;
  CDS_MovimentosVLRTOTALMOV.AsFloat := CalculaFormula('001');
end;

procedure TdmDados.CDS_ItensMovimentosAfterPost(DataSet: TDataSet);
begin
  CDS_ItensMovimentos.ApplyUpdates(-1);
end;

procedure TdmDados.CDS_MovimentosAfterPost(DataSet: TDataSet);
begin
  CDS_Movimentos.ApplyUpdates(-1);
end;
end.
```

Listagem 4. Código SQL do SQL_Calculos

```
SELECT
  SUM(I.VLROPCIONAL1) SVLROPCIONAL1,
  SUM(I.VLROPCIONAL2) SVLROPCIONAL2,
  SUM(I.VLRICMS) SVLRICMS,
  SUM(I.VLRIPI) SVLRIPI,
  SUM(I.VLRTOTALITEM) SVLRTOTALITEM
FROM ITENSMOVIMENTOS I
WHERE IDMOV=:PIDMOV
```

Importante: Nos *DataSetProviders*, propriedade *Options* a opção *AllowCommandText* deverá ter o valor *True*, para que possamos colocar as instruções SQL nas *ClientDataSets* (adicione para cada componente o respectivo código SQL das tabelas do banco, ex: "select * from FORMULAS" no *CDS_Formulas*). Adicione também os *TFields* em todos os *ClientDataSets*.

Para os formulários da aplicação, criaremos um formulário básico de cadastro, que chamaremos de "frmCadBase" (salve-o

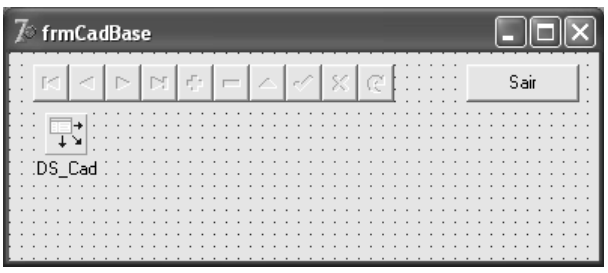


Figura 3. Formulário básico de cadastro contendo o código comum a todos os cadastros

ClubeDelphi 76.indb 17 27.07.06 15:38:51

como "ufrmCadBase.pas", do qual os outros serão descendentes). Para isso, vamos criar um novo formulário e o deixaremos parecido com o mostrado na **Figura 3 (**o *DBNavigator* deverá ter sua propriedade *DataSource* setada para *DS_Cad*).

O código, referente ao evento *OnClick* do botão e *OnShow* do formulário, está na **Listagem 5.**

Listagem 5. Eventos do botão e formulário

```
procedure TfrmCadBase.FormShow(Sender: TObject);
begin
  if not DS_Cad.DataSet.Active then
    DS_Cad.DataSet.Open;
end;
procedure TfrmCadBase.btnSairClick(Sender: TObject);
begin
  Close;
end;
end.
```

Através da técnica de herança visual de formulários, criaremos os demais formulários da nossa aplicação, para isso, acesse o menu *File>New>Other*, escolha na aba *P_Exemplo* o item *frmCadBase* e clique em OK.

Crie dois formulários, chamando-os de: "frmCadFormulas" e "frmCadMovimentos" (salvando-os respectivamente como "ufrmCadFormulas" e "ufrmCadMovimentos"). O formulário de cadastro de Fórmulas deve ficar semelhante ao mostrado na **Figura 4.**

Os itens mostrados nas fórmulas disponíveis, conforme descrições mostradas na **Figura 5**, dividem-se em três categorias:

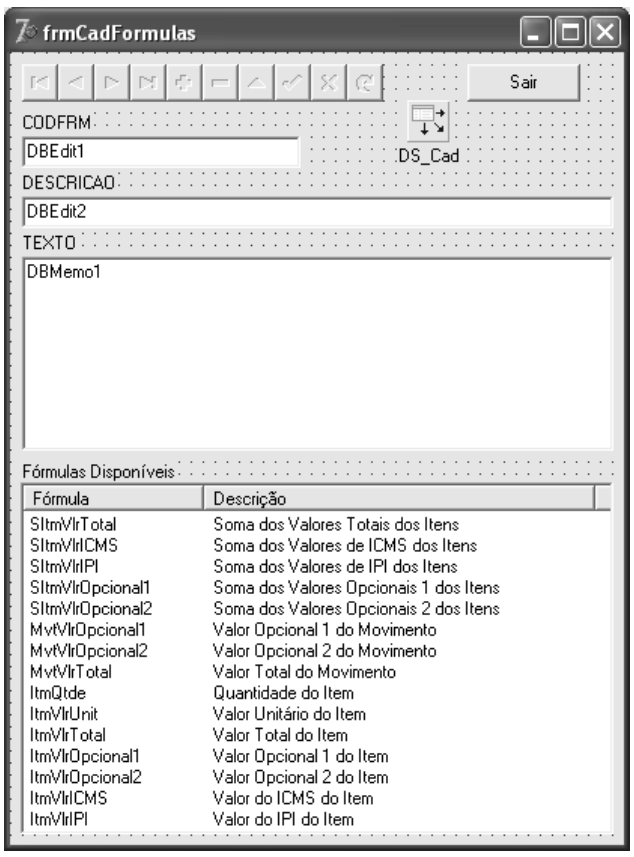


Figura 4. Formulário de cadastro das fórmulas

- Campos que trazem a soma de um determinado campo dos *Itens Movimentos*:
  - *SItmVlrTotal*, *SItmVlrICM*, *SItmVlrIPI*, *SItmVlrOpcional1* e *SItmVlrOpcional2*
- Campos que trazem o valor de um determinado campo dos *Movimentos*:
  - *MvtVlrOpcional1*, *MvtVlrOpcional2* e *MvtVlrTotal*
- Campos que trazem o valor de um determinado campo dos Itens Movimentos:
  - *ItmQtde*, *ItmVlrUnit*, *ItmVlrTotal*, *ItmVlrOpcional1*, *ItmVlrOpcional2*, *ItmVlrICMS* e *ItmVlrIPI*

O formulário de Movimentos ficará semelhante a **Figura 5.**

No formulário de Movimentos, no botão *Atualizar Total* chamaremos simplesmente os métodos *Edit* e *Post* da *CDS_Movimentos*, ou *DS_Cad.DataSet*, visto que o mesmo está ligado a esse *DataSource*.

Essa chamada é feita somente para que o evento *BeforePost* da *CDS_Movimentos* seja executado, pois ele é o responsável direto pela atualização dos valores dos itens e do movimento, como podemos observar na **Listagem 3**, com os métodos e eventos criados no *dmDados*.

A **Listagem 6**, mostra o código a ser adicionado no *Atualizar Total* e evento *OnShow* do formulário.

Listagem 6. Código do frmCadMovimentos

```
...
uses udmDados;
...
procedure TfrmCadMovimentos.FormShow(Sender: TObject);
begin
  inherited;
  { Abrir tabela-filha }
  if not DS_Itens.DataSet.Active then
    DS_Itens.DataSet.Open;
end;
procedure TfrmCadMovimentos.Button1Click(
  Sender: TObject);
begin
  inherited;
  { Somente chamar Edit e Post para que ele refaça
    os cálculos do total do movimento }
  DS_Cad.DataSet.Edit;
  DS_Cad.DataSet.Post;
end;
end.
```

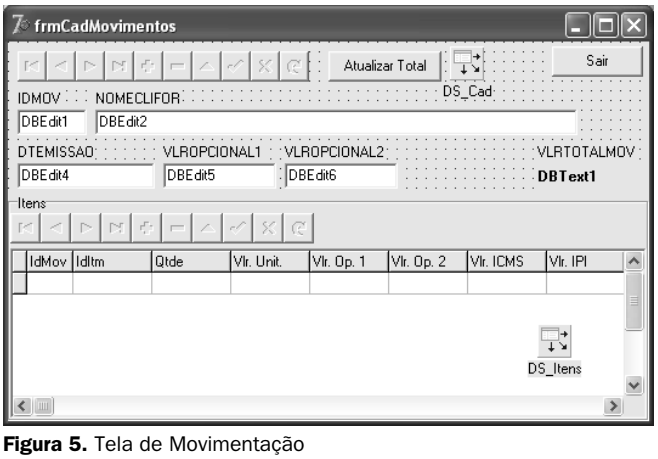


Figura 5. Tela de Movimentação

ClubeDelphi 76.indb 18 27.07.06 15:38:51

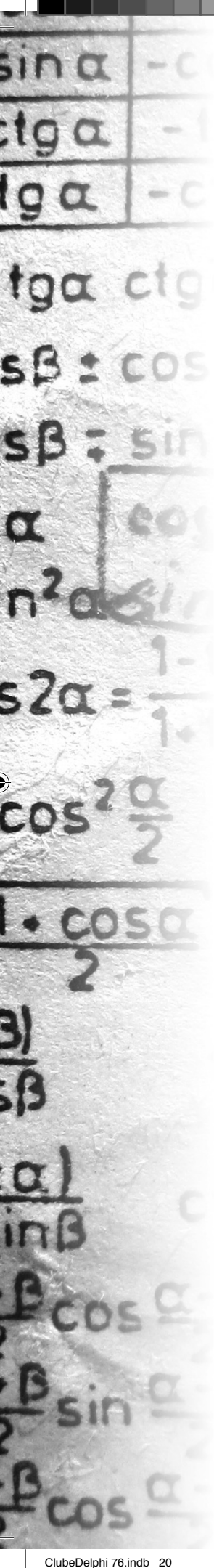
Codificando a aplicação

Agora que as telas de nossa aplicação estão criadas e conhecemos a classe *TEvaluator*, vamos codificar nosso projeto para que os valores totais sejam calculados. Criaremos uma unit chamada "U_FuncoesFormulas.pas" (*File>New>Unit*) e digitaremos o código conforme a **Listagem 7**, que conterá as funções chamadas na fórmula.

Listagem 7. Unit U_FuncoesFormulas

```
U_FuncoesFormulas;

interface

uses Dialogs, SysUtils;

  { Exemplo de TFloatFunc }
  function SomaItmVlrTotal: Double;
  function SomaItmVlrICMS: Double;
  function SomaItmVlrIPI: Double;
  { Exemplo de TUnaryFunc }
  function SomaItmVlrOpcional1(Movimento: Double): Double;
  function SomaItmVlrOpcional2(Movimento: Double): Double;

implementation

uses udmDados;
function SomaItmVlrTotal: Double;
begin
  with dmDados.SQL_Calculos do
  begin
    if Active then
      Close;
    ParamByName('PIDMOV').AsInteger :=
      dmDados.CDS_MovimentosIDMOV.AsInteger;
    Open;
    Result := FieldByName('SVLRTOTALITEM').AsFloat;
  end;
end;

function SomaItmVlrICMS: Double;
begin
  with dmDados.SQL_Calculos do
  begin
    if Active then
      Close;
    ParamByName('PIDMOV').AsInteger :=
      dmDados.CDS_MovimentosIDMOV.AsInteger;
    Open;
    Result := FieldByName('SVLRICMS').AsFloat;
  end;
end;
function SomaItmVlrIPI: Double;
begin
  with dmDados.SQL_Calculos do
  begin
    if Active then
      Close;
   ParamByName('PIDMOV').AsInteger := dmDados.CDS_MovimentosIDMOV.AsInteger;
    Open;
    Result := FieldByName('SVLRIPI').AsFloat;
  end;
end;

function SomaItmVlrOpcional1(
  Movimento: Double): Double;
begin
  with dmDados.SQL_Calculos do
  begin
    if Active then
      Close;
    ParamByName('PIDMOV').AsFloat := Movimento;
    Open;
    Result := FieldByName('SVLROPCIONAL1').AsFloat;
  end;
end;
function SomaItmVlrOpcional2(Movimento: Double): Double;
begin
  with dmDados.SQL_Calculos do
  begin
    if Active then
      Close;
    ParamByName('PIDMOV').AsFloat := Movimento;
    Open;
    Result := FieldByName('SVLROPCIONAL2').AsFloat;
  end;
end;
end.
```

ClubeDelphi 76.indb 19 27.07.06 15:38:52



O código da listagem anterior utilizará o *SQL_Calculos*, pois retornará individualmente os valores trazidos pela SQL configurado nesse componente. Por exemplo: a função *SomaItmVlrOpcional2* passa como parâmetro para o *SQL_Calculos* a variável *Movimento*, retornando o campo *SVLROPCIONAL2*.

Para uma melhor compreensão do uso da classe *TEvaluator* foram criadas cinco funções que retornam a soma de valores dos itens, sendo duas do tipo *TFloatFunc* e três *TUnaryFunc*. Além das funções foram criadas variáveis que recebem o valor de campos da tabela de Movimentos ou Itens no registro corrente.

Agora, a parte principal de nosso código está concentrada na função *CalculaFormula*, que apresenta uma sintaxe básica para o uso da classe *TEvaluator* que é:

1) Criação do objeto *Evaluator*;
2) Adição das variáveis e funções ao avaliador;
3) Passagem dos valores às variáveis adicionadas ao avaliador;
4) Execução da fórmula;
5) Tratamento de erros na execução e
6) Destruição do objeto *Evaluator*.

Podemos observar que somente as variáveis devem ter seus valores passados ao objeto avaliador de expressões, pois as funções que adicionamos a ele vão ser chamadas automaticamente ao executarmos a fórmula, não necessitando serem chamadas dentro da função.

Para tratarmos possíveis erros de execução usamos o bloco *try..except..end* fazendo com que, caso ocorra exceção, a fórmula simplesmente retorne valor nulo (Result := 0) e avise sobre a ocorrência do erro numa mensagem personalizada.

E como temos um objeto criado em *runtime*, usamos o bloco *try..finally..end* pois queremos, tendo sido executada a função com sucesso ou não, o objeto seja destruído, para evitarmos erros de memória.

Testando a aplicação

Para o teste de nossa aplicação vamos primeiramente cadastrar nossas duas fórmulas envolvidas nos cálculos (na ordem os campos: CODFRM, DESCRICAO e TEXTO):

001 - Valor Total do Movimento – (SItmVlrTotal + SItmVlrIPI) – MvtVlrOpcional1 e

002 - Valor Total do Item - (ItmQtde * ItmVlrUnit) + ItmVlrOpcional1

E, logo após, vamos cadastrar um movimento, para isso, no formulário de Movimentos, pelo *DBNavigator* crie um novo registro, preencha os campos do movimento, salve e só então crie seus itens pelo *DBNavigator* do *GroupBox* preenchendo seus campos.

Não esqueça de salvar novamente o movimento após a inserção dos itens para calcular seu total. Para certificar do funcionamento do sistema, basta alterar uma das fórmulas e salvar o movimento para que os totais sejam recalculados.

Classes de avaliadores de expressões

Na unit *JclExprEval*, temos outro avaliador de expressões, o *TCompiledEvaluator*. Fora da biblioteca do JEDI podemos citar também alguns componentes disponíveis com a mesma funcionalidade, entre eles: *TCioinaEval*, *TCioinaComplexEval*, *StExpr*, *CalcExpress*, *Parser*, *ExpressionParser* e *RPNParser*.

Temos também um artigo publicado no site da CloubeDelphi que mostra como construir um componente analisador de expressões. E, além disso, para encontrar mais analisadores de expressão, podemos recorrer ao nosso bom e velho Google.

Conclusão

Pudemos ver neste exemplo, através de soluções fáceis de serem implementadas, como podemos agregar valor ao nosso software de modo que ele se torne muito mais atrativo aos olhos de nossos clientes e nos possibilitem um diferencial competitivo.

Com base nisso poderíamos inclusive, possibilitar ao usuário escolher quais fórmulas utilizar através de uma parametrização nos tipos de movimento, ou seja, orçamentos, pedidos etc. flexibilizando assim mais nosso aplicativo. ■

Links

Site de pesquisa de código-fonte de projetos, bibliotecas e outros

www.koders.com

Site do projeto JEDI

sourceforge.net/projects/jcl/

Site do TCioinaComplexEval

www.geocities.com/alexei_cioina/cioinaeval.html

Site do RPNParser 0.2b

www.joaomorais.com.br/pascal/download.php?cat=101

Site da Borland Developer Network com download de avaliadores de expressão

cc.borland.com/Item.aspx?id=22972

O que é precioso você não entrega nas mãos de qualquer um. Por isso, quando pensar em Internet Data Center, procure a líder em número de clientes no Rio. Procure a ALOG. Utilizamos as mesmas estruturas da antiga filial carioca da .comDominio, o que lhe garante uma superestrutura em um prédio-cofre com sistemas de controle de acesso, de detecção e combate a incêndio com gás FM200, ambiente climatizado, no breaks e geração própria de energia de 1200KVA's, infra-estrutura de redes e acesso redundantes com monitoramento 24x7. Tudo para atender aos sistemas de missão crítica dos nossos clientes. Mas não é só isso. A ALOG ainda dispõe de atendimento personalizado, eficiente e ágil, com capacidade para atender aos nossos mais de 240 clientes corporativos com rapidez.
Saiba tudo o que podemos fazer pela sua empresa. Ligue para nós, (21) 3083-3333.

ALOG, SEUS DADOS BEM-CUIDADOS.

Controle de incêndio com FM200

1200KVA's de geração própria

Controle de acesso

Portas blindadas

ALOG
data centers
www.alog.com.br

CRISTIANO CAETANO

CVS – Controle de versão

Conheça uma ferramenta free e integrada ao Windows Explorer

CRISTIANO CAETANO
(*c_caetano@hotmail.com*)
autor do livro CVS: Controle de Versões e Desenvolvimento Colaborativo de Software. Com 10 anos de experiência, já trabalhou na área de qualidade e teste de software para grandes empresas como Zero G e DELL. Atualmente está exercendo a função de Software Test Engineer na HP Invent. O autor também pode ser contactado no seguinte endereço: spaces.msn.com/softwarequality.

O CVS é uma ferramenta *open source* que implementa as principais funções pertinentes ao processo de controle de versões. Basicamente, o CVS armazena em seu repositório as modificações realizadas num arquivo ao longo do tempo; cada modificação é identificada por um número chamado *Revisão*.

Toda *Revisão* armazena as modificações realizadas, quem realizou as modificações, quando foram realizadas, entre outras informações. Além disso, o CVS conta com um mecanismo capaz de controlar os acessos simultâneos e as modificações paralelas, garantindo a integridade das modificações e a atomicidade das operações.

De maneira geral, o fluxo de trabalho básico do CVS consiste em retirar os arquivos de um projeto para a área de trabalho, realizar as modificações e, por último, transferir as modificações ao repositório, como pode ser visto na **Figura 1.**

No entanto, durante o curso das modificações, você poderá transferir as suas modificações ao repositório ou obter as modificações realizadas por outras pessoas. O completo entendimento dos termos apresentados na **Tabela 1** é fundamental para auxiliar a leitura dos próximos tópicos.

TortoiseCVS

O TortoiseCVS é um *front end* gráfico que torna o uso do CVS mais fácil e intuitivo. Basicamente, o TortoiseCVS realiza as principais operações do CVS por meio de menus de contexto integrados ao Windows Explorer, como pode ser visto na **Figura 2.**

O TortoiseCVS é, em virtude da sua proposta, uma ferramenta extremamente eficiente cuja principal característica é a sua independência da linguagem de programação. Em razão disso, você poderá utilizar o TortoiseCVS sem precisar inserir nenhum *plugin* no IDE do Delphi. Dessa

ClubeDelphi 76.indb 22 27.07.06 15:38:59

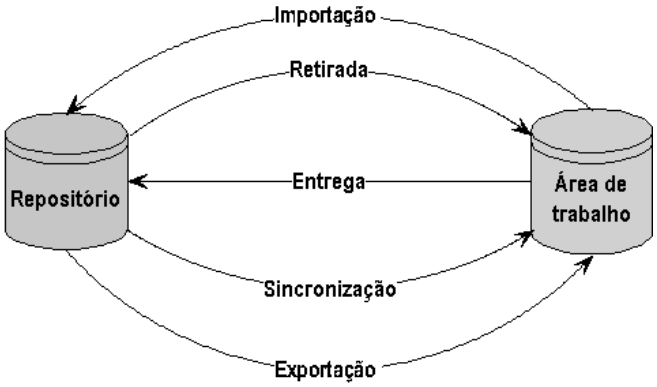


Figura 1. Operações realizadas pelo CVS

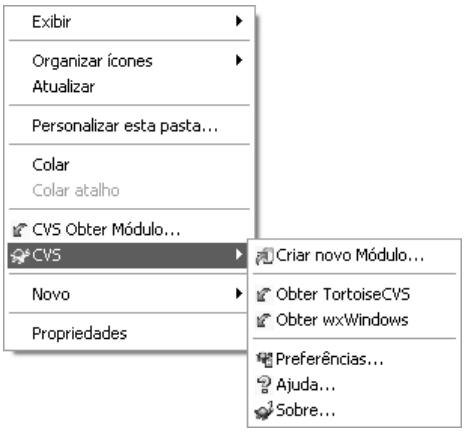


Figura 2. Menus de contexto integrados ao Windows Explorer

Termo	Descrição
Workspace, SandBox (Área de trabalho)	Termo usado para representar um diretório no computador cliente onde os arquivos de um projeto serão transferidos durante uma retirada.
Repository (Repositório)	Local onde os arquivos submetidos ao controle de versões são armazenados.
Revision (Revisão)	Termo usado para descrever a numeração atribuída pelo CVS a cada modificação de um arquivo.
Release, Version (Liberação)	Termo usado para descrever a denominação atribuída a um conjunto de arquivos para identificar determinado ponto no tempo, sobretudo, quando se quer identificar um conjunto de novas características ou correções de um software.
Commit (Entregar, Submeter)	Termo usado para descrever a transferência de um ou mais arquivos modificados da Área de Trabalho para um Repositório.
Checkout (Retirar, Recuperar)	Termo usado para descrever a retirada de um ou mais arquivos de um repositório para a área de trabalho.
Diff (Comparação das diferenças)	Termo usado para descrever o processo de comparação das modificações entre revisões diferentes de um arquivo.
Import (Importação)	Termo usado para descrever o processo de transferência dos arquivos de um projeto a um repositório, dando início ao controle de versões.
Module (Projeto, Módulo)	Termo usado para descrever um nome simbólico associado a um conjunto de diretórios e arquivos que compõem um projeto, a fim de facilitar o acesso e a manipulação.
Branch (Ramo)	Termo usado para descrever o processo de divisão dos arquivos de um projeto em linhas de desenvolvimento independentes.

Tabela 1. Termos utilizados no controle de versões

forma, como descrito anteriormente, todas as operações serão realizadas por meio de menus de contexto do Windows Explorer.

O TortoiseCVS é uma ferramenta gratuita distribuída sob a licença GPL. Segundo o site do desenvolvedor, ele roda normalmente nas seguintes versões do Windows: 95, 98, ME, NT, 2000 e XP.

> **Dica**: Na edição 73 também apresentamos uma outra alternativa gratuita para controle de versão, confira.

Repositório

Basicamente, um repositório é um diretório que tem por objetivo abrigar todos os arquivos de um projeto sob o controle de versões. O repositório poderá ser criado num diretório compartilhado no seu computador ou num servidor (Windows ou Linux). Naturalmente, quando o repositório é criado num servidor, você deverá determinar um método de acesso ao repositório para garantir a segurança dos arquivos sob o controle de versões.

Sendo assim, para fins didáticos, os exemplos apresentados nas seções seguintes serão realizados num repositório local. Para tal tarefa, você deverá criar e compartilhar um diretório chamado "CVSROOT" na raiz do seu computador (C:\CVSROOT).

Importação

Uma vez que o repositório tenha sido criado, o primeiro passo para iniciar o controle de versões é chamado de importação. Nesse passo, os arquivos da sua aplicação Delphi são transferidos para o repositório e submetidos ao controle de versões do CVS. Para tal tarefa, devemos copiar os arquivos do projeto numa área temporária a fim de realizarmos a importação dos arquivos. Para fins de análise e entendimento, vamos transferir os arquivos de um suposto projeto de um software chamado *Calculadora* para o diretório "C:\Temp".

ClubeDelphi 76.indb 23 27.07.06 15:39:01

A importação do projeto deverá ser realizada por meio do menu de contexto *Criar Novo Módulo*. Na caixa de diálogo, o TortoiseCVS fornece opções para acessar repositórios remotos por meio de qualquer método de autenticação suportado pelo CVS. Mas, no entanto, como havíamos definido anteriormente, todos os exemplos serão realizados utilizando um repositório local. Em razão disso, você deverá selecionar a opção *Diretório Local (:local:)* do item *Protocolo*.

O campo *Pasta do Repositório* deverá ser preenchido com "C:\CVSROOT" que é a localização do repositório criado anteriormente na raiz do seu computador. O campo *Módulo* indica um apelido que será utilizado pelo TortoiseCVS todas as vezes que ele se referir ao nosso projeto e deverá ser preenchido com o texto "CALC", como pode ser visto na **Figura 3**.

Após confirmar os dados, o TortoiseCVS exibirá um diálogo informando que não existe nenhum repositório criado no diretório informado. Você deverá selecionar a opção para inicializar um novo repositório, como pode ser observado na **Figura 4**.

Esse diálogo será exibido somente na primeira vez que você importar o projeto em função de que o repositório não tinha sido inicializado ainda. Posteriormente, assim que a criação do novo módulo e a inicialização do repositório forem completadas, você deverá adicionar os arquivos a esse módulo recém criado. Para tal tarefa, você deverá acessar o menu de contexto *CVS Adicionar Conteúdo*.

O TortoiseCVS exibirá um diálogo solicitando a confirmação dos arquivos que serão adicionados. Adicionalmente, você poderá informar o formato do arquivo, o tipo de final de linha do arquivo, entre outras opções, como pode ser visto na **Figura 5**.

Por fim, devemos realizar a operação de entrega, também chamada de *Commit* para transferir os arquivos do projeto definitivamente para o repositório do CVS. Essa operação deverá ser realizada através do menu de contexto *CVS Gravar*

O TortoiseCVS exibirá um diálogo solicitando a confirmação dos arquivos que serão transferidos para o repositório. Neste diálogo, você também deverá escrever um breve comentário descrevendo as modificações realizadas. Além disso, antes de confirmar a operação você poderá modificar o formato do arquivo, comparar as suas modificações com outras versões armazenadas no repositório, entre outras opções, como pode ser observado na **Figura 6.**

Retirada dos arquivos do repositório

Assim que os arquivos de um projeto forem importados com sucesso, podemos finalmente iniciar as operações de controle de versões. Para tal tarefa, devemos retirar os arquivos armazenados no repositório e transferi-los para a área de trabalho, onde poderemos realizar as modificações nos arquivos do projeto CALC.

A área de trabalho é um diretório qualquer no disco rígido. Nesse caso, vamos assumir que a nossa área de traba-

Figura 3. Caixa de diálogo para criar um novo módulo

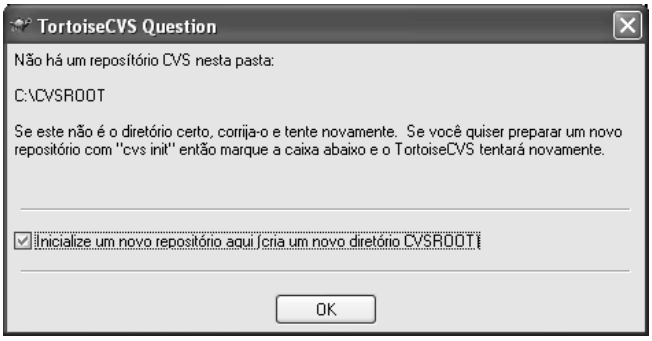


Figura 4. Inicialização do repositório

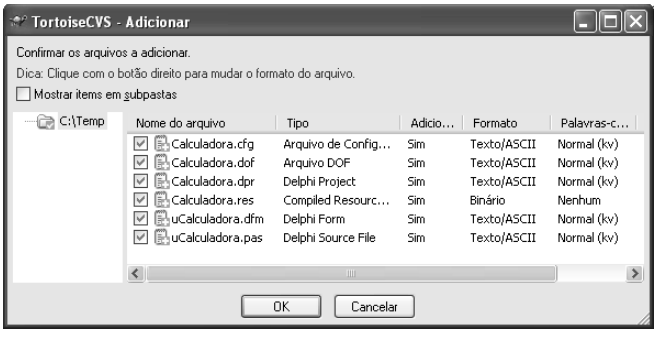


Figura 5. Adicionando arquivos ao módulo

ClubeDelphi 76.indb 24 27.07.06 15:39:02

lho será o diretório "C:\Projetos". Na prática, para realizar essa tarefa, você deverá clicar sobre o diretório (*Projetos*) e acessar o menu de contexto *CVS Obter Módulo*. Como resultado, o TortoiseCVS exibirá o diálogo para realizar a retirada dos arquivos, também conhecido como *Checkout*.

Por meio desse diálogo, você poderá configurar onde está localizado o repositório de arquivos, qual projeto você deseja retirar, entre outras opções, como pode ser visto na **Figura 7.**

Cabe lembrar que a não ser que você faça alguma configuração adicional, o TortoiseCVS armazena uma lista dos projetos/repositórios acessados anteriormente, facilitando assim posteriores retiradas. Por fim, assim que a operação de retirada for concluída, você poderá começar a trabalhar com os arquivos do projeto CALC que foram transferidos para o diretório *Projetos*.

Modificando os arquivos

Assim que o projeto for transferido para a área de trabalho, poderemos iniciar a edição dos arquivos. Perceba que após a importação e durante a edição dos arquivos, o TortoiseCVS modifica os ícones dos arquivos a fim de dar um significado visual sobre o seu status em relação ao repositório, como pode ser visto na **Figura 8.**

A rigor, a edição dos arquivos do seu projeto não mudará em nada. Utilize o Delphi da mesma maneira que você está acostumado. No entanto, para que a adição e remoção de arquivos sejam entendidas pelo CVS, você deverá selecionar o arquivo desejado e acessar os menus de contexto *CVS Adicionar* ou *CVS Remover* como mostrado na **Figura 9.**

Por fim, devemos lembrar de realizar a operação de entrega (*Commit*) ao final do nosso trabalho para garantir que todas as modificações, adições e remoções sejam definitivamente refletidas no repositório do CVS.

Configurações Adicionais

O TortoiseCVS em virtude da sua arquitetura, ou melhor, da arquitetura do CVS, adiciona automaticamente um *linefeed* (#10) antes de cada caractere *carriage return* (#13) dentro dos arquivos armazenados em seu repositório. Na verdade, ele só realiza esse tipo de operação em arquivos definidos com o formato texto puro (*plain text*).

O problema é que ele identifica por default os arquivos DFM como se fossem arquivos texto puro, corrompendo os mesmos durante o armazenamento desses arquivos no repositório CVS. Para sanar esse problema, você deverá selecionar individualmente cada arquivo DFM e mudar o campo *Formato* de *Text/ASCII" para Binary* quando realizar a operação de entrega (*Commit*) do arquivo.

Como esse procedimento é trabalhoso e sujeito a erros, você poderá configurar o TortoiseCVS de modo que ele possa automaticamente identificar todos os arquivos DFM como se fossem arquivos binários. Em resumo, existe um arquivo de configuração utilizado pelo TortoiseCVS onde poderão ser definidos os formatos dos arquivos conforme a sua extensão ou por uma expressão regular.

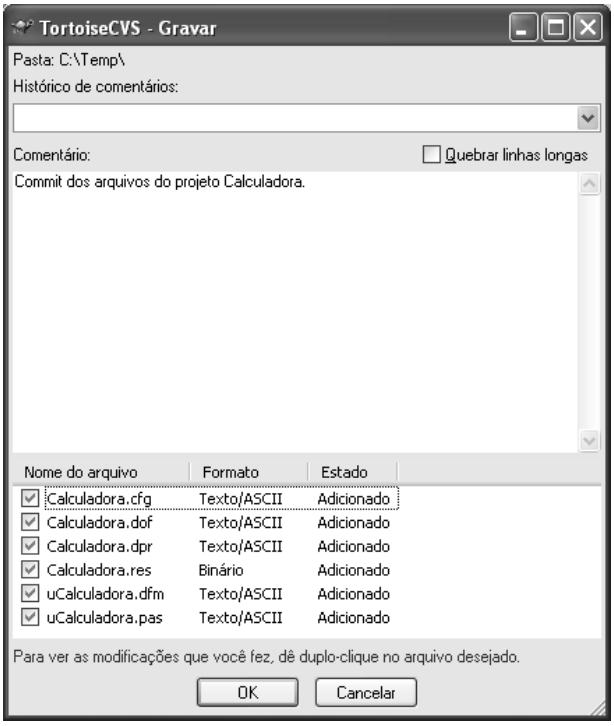


Figura 6. Transferência definitiva dos arquivos ao repositório CVS

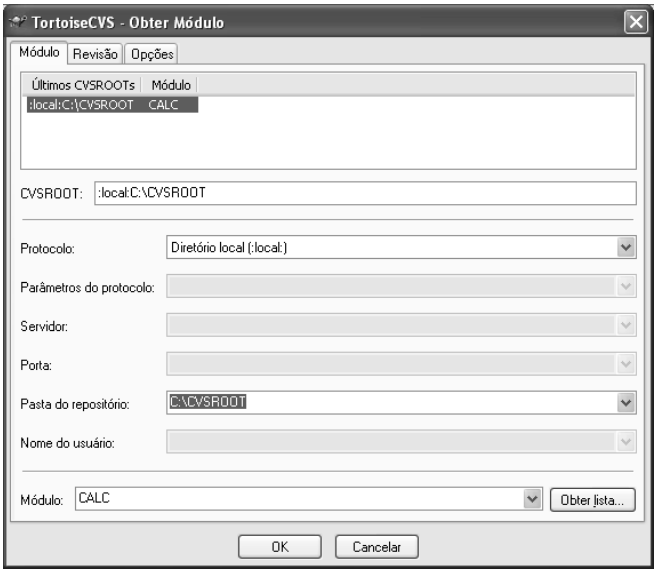


Figura 7. Retirada dos arquivos do repositório CS

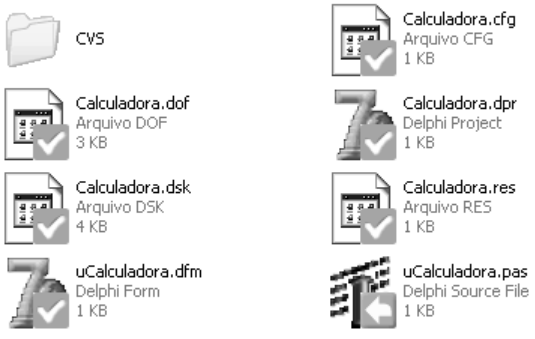


Figura 8. Ícones modificados indicando o status atual dos arquivos

ClubeDelphi 76.indb 25 27.07.06 15:39:02

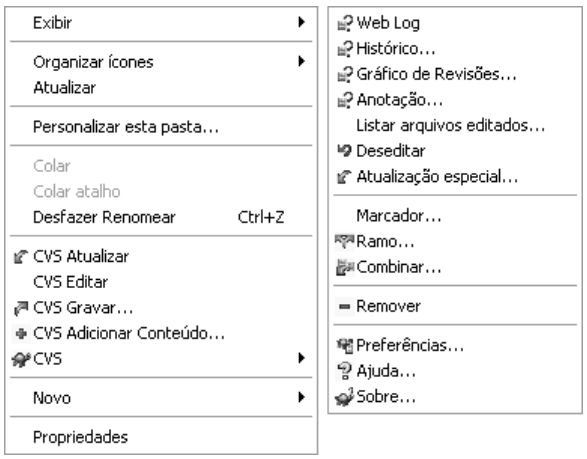


Figura 9. Adicionando e removendo arquivos

Para realizar tal tarefa, você deverá editar o arquivo *TortoiseCVS.FileTypes* (utilizando o Bloco de Notas) que está localizado na raiz da instalação do TortoiseCVS (normalmente é o diretório C:\Arquivos de programas\TortoiseCVS) e inserir a expressão "*.dfm binary" no final do arquivo, como pode ser visto na **Figura 10.**

Normalmente arquivos executáveis e arquivos DCU não deverão ser submetidos ao controle de versões. Via de regra, arquivos binários resultantes da compilação, normalmente não deverão ser submetidos ao controle de versões.

Em razão disso, antes de realizar a importação ou adição de arquivos, você deverá configurar o TortoiseCVS para ignorar os arquivos que não deverão ser submetidos ao controle de versões. Para tal tarefa, você deverá acessar a guia *Arquivos Ignorados* do diálogo *Preferências* e adicionar os arquivos que deverão ser ignorados, como pode ser visto no exemplo descrito na **Figura 11.**

Conclusão

Neste artigo foram apresentadas alguns dos recursos oferecidos pelo TortoiseCVS. Com base nessas informações você já estará apto para iniciar o controle de versões dos seus projetos, aumentando assim a confiabilidade e a rastreabilidade do seu trabalho. Se você desejar conhecer os demais recursos oferecidos pelo CVS, vale a pena dar uma lida no livro CVS: Controle de Versões e Desenvolvimento Colaborativo de Software (veja a seção *Links*). ■

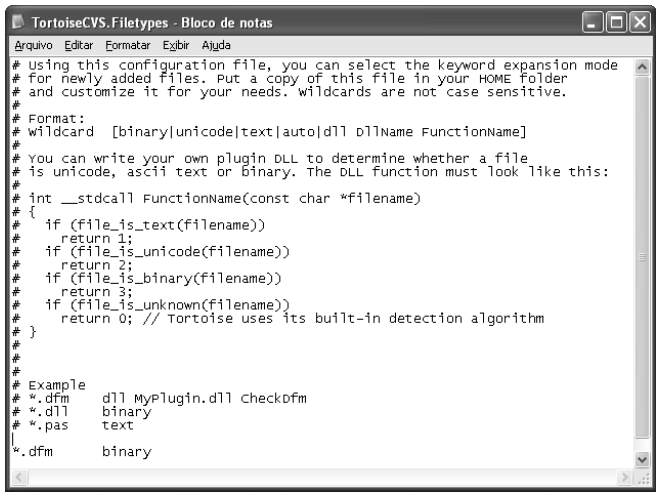


Figura 10. Definindo automaticamente o formato dos arquivos DFM

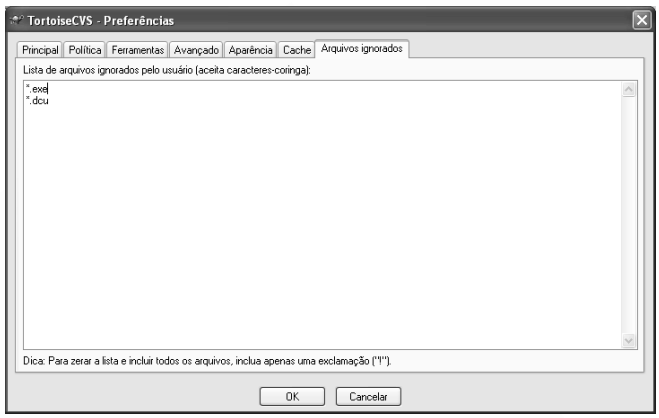


Figura 11. Definindo os arquivos que deverão ser ignorados

Links

TortoiseCVS homepage

www.tortoisecvs.org

Resolving DFM corruption issues with CVS

bdn.borland.com/article/0,1410,26068,00.html

CVS: Controle de Versões e Desenvolvimento Colaborativo de Software

www.novateceditora.com.br/livros/cvs

Informações sobre o CVS

ximbiot.com/cvs/wiki/index.php?title=Main_Page

Blog do autor contendo informações adicionais sobre CVS e qualidade de software

spaces.msn.com/softwarequality

Edições Anteriores
Complete a sua Coleção.

EDIÇÃO 51

- Novidades da Delphi Language: Parte IV
- Database Explorer
- POO na Prática
- Criando controles ASP.NET

EDIÇÃO 52

- Novidades da Delphi Language: Parte IV
- Database Explorer
- POO na Prática
- Criando controles ASP.NET

EDIÇÃO 53

- Novidades da Delphi Language: Parte IV
- Database Explorer
- POO na Prática
- Criando controles ASP.NET

EDIÇÃO 54

- Carga dinâmica de pacotes
- Platarforma .NET
- Segredos da VCL
- Threads e Concexões

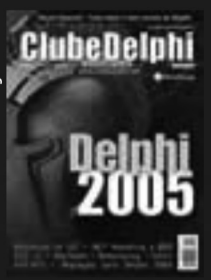
EDIÇÃO 55

- Web Services
- Guitar Explorer
- Rave Language
- Crystal Reports

EDIÇÃO 56

- Novidades da Delphi Language: Parte IV
- Database Explorer
- POO na Prática
- Criando controles ASP.NET

EDIÇÃO 57

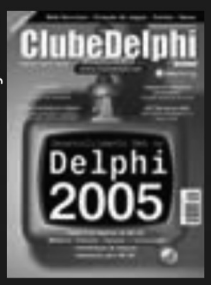


- Novidades da Delphi Language: Parte IV
- Database Explorer
- POO na Prática
- Criando controles ASP.NET

EDIÇÃO 58

- Novidades da Delphi Language: Parte IV
- Database Explorer
- POO na Prática
- Criando controles ASP.NET

EDIÇÃO 59



- Novidades da Delphi Language: Parte IV
- Database Explorer
- POO na Prática
- Criando controles ASP.NET

EDIÇÃO 60

- IBExpert Total
- Migrando para Delphi for .NET Parte I
- StarTeam passo a passo
- TestComplete, Testes de aplicações

EDIÇÃO 61

- Programação Orientada por Objetos
- Acesso ao InterBase e Firebird
- Crystal Reports, Relatórios mestre
- Parâmetros em relatórios Rave

EDIÇÃO 62

- TreeView com Dados
- Criação de Jogos com o Delphi Parte V
- Curso de ASP.NET com Delphi Parte VII
- CMMI, Quem precisa dele?

EDIÇÃO 63

- Sistema SysCash
- Longhorn, O Futuro do Windows
- Usando o Repeater e DataList
- O que é CodeDOM?

EDIÇÃO 64

- DevExpress QuantumGrid
- Structured Query Language
- Guia de referência Parte II
- Comandos DML; Crystal Reports

EDIÇÃO 66

- Novidades da Delphi Language: Parte IV
- Database Explorer
- POO na Prática
- Criando controles ASP.NET

EDIÇÃO 67

- Novidades da Delphi Language Parte III
- Namespaces
- Comandos DCL e views
- DataSnap e DCOM + Parte III

EDIÇÃO 68

- Novidades da Delphi Language: Parte IV
- Database Explorer
- POO na Prática
- Criando controles ASP.NET

EDIÇÃO 69

- Delphi 2006
- VCL - Novos Componentes
- Desenvolvimento com C++
- Padrões de Projeto e POO

ClubeDelphi

DevMedia GROUP

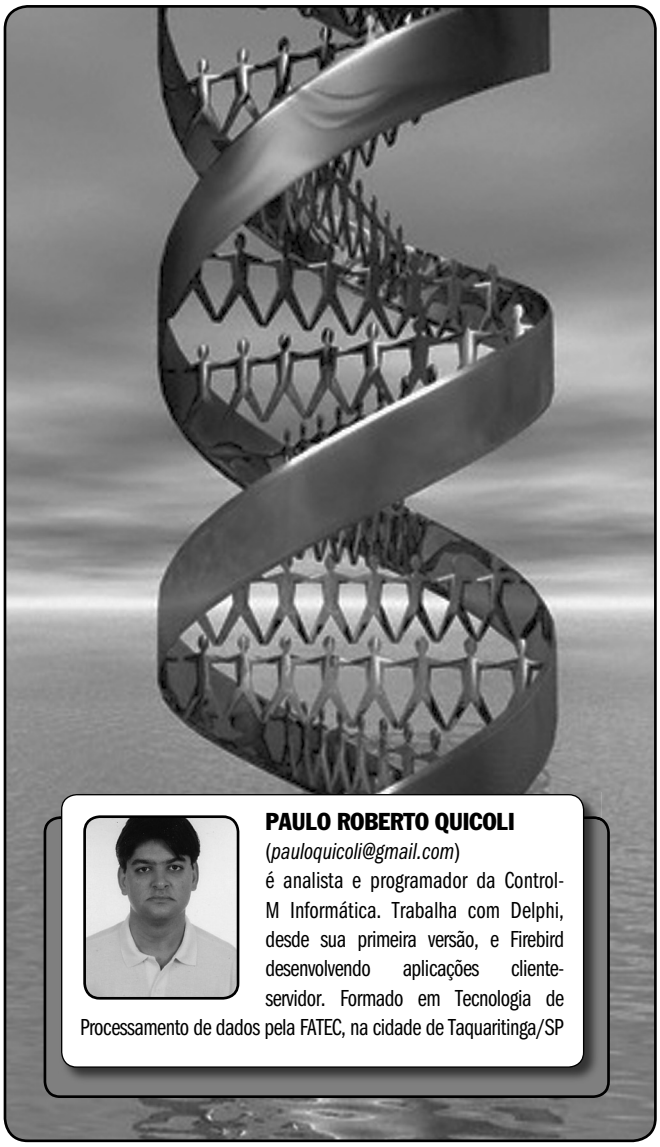
Acesse agora a nossa página de compra ou se preferir, entre em contato com a nossa Central de Atendimento através do telefone: (21) 2283-9012.

www.devmedia.com.br/anteriores

PAULO QUICOLI

Mini-Curso de Delphi e UML

Construa uma aplicação completa usando análise UML, POO e Persistência de Objetos – Parte I



PAULO ROBERTO QUICOLI
(*pauloquicoli@gmail.com*)
é analista e programador da Control-M Informática. Trabalha com Delphi, desde sua primeira versão, e Firebird desenvolvendo aplicações cliente-servidor. Formado em Tecnologia de Processamento de dados pela FATEC, na cidade de Taquaritinga/SP

Muitos desenvolvedores iniciantes em programação Orientada a Objetos ouvem seus conceitos e vantagens, aliás, hoje só se fala nisso, então podemos dividi-los, no geral, em dois grupos: dos que acham que "não precisam disso" e dos "entusiasmados". Infelizmente os participantes dos dois grupos tendem, em sua maioria, a ter o mesmo fim: acabam por não utilizar a Orientação a Objetos na prática, em uma aplicação real.

Cada grupo citado reage de uma maneira diferente, talvez o primeiro possa achar que suas aplicações não virão a se beneficiar da Orientação a Objetos ou na pior das hipóteses acham que tudo isso é apenas uma onda, algo que vem e que passa.

O outro grupo, no entanto, deseja desenvolver o assunto, mas por falta de exemplos didáticos, por falta de alguma teoria ou até mesmo por não conhecer ferramentas auxiliadoras, acabam não tendo sucesso em seus empreendimentos, por achar o desenvolvimento muito lento ou muito complicado.

Este mini-curso não tem a pretensão de ditar regras, e sim, compartilhar experiências e oferecer aos desenvolvedores que se enquadram nos grupos citados anteriormente ou em qualquer outro, uma base de conhecimento, simples, didático e funcional para que possa servir como base para projetos mais audaciosos.

Vamos desenvolver uma aplicação de fácil entendimento de todos, passando pela fase de análise à implementação.

Sistema exemplo

Como sou um colecionador das revistas do grupo DevMedia, em especial a ClubeDelphi e agora a MSDN Magazine, vamos desenvolver um catálogo eletrônico para elas. Isso é muito útil porque com o passar do tempo esquecemos os artigos de cada revista publicada e quando precisamos pes-

ClubeDelphi 76.indb 28 27.07.06 15:39:05

quisar sobre algum exemplo, nem lembramos que talvez o assunto já tenha sido abordado em alguma edição. Com esse catálogo, poderemos lançar os artigos, consultá-los e emitir algum relatório do acervo ou relatório estatístico.

Começando a analisar

Em relação a extrair informações dos clientes para implementação de um sistema, a UML oferece um recurso chamado Caso de Uso (*Use Case*). Um Caso de Uso é uma ferramenta que será utilizada durante todo o desenvolvimento do sistema, isso porque ela descreve as várias opções de realização de uma determinada funcionalidade do sistema, sempre no olhar do usuário.

Descrevendo um Caso de Uso

Descrever um Caso de Uso significa relatar a forma com que o usuário deseja que determinada operação seja realizada, para que depois sirva como base para o **diagrama de caso de uso**. Existem várias formas de se descrever um Caso de Uso, embora todas sejam similares, aqui vou apresentar a de minha preferência. Ao obter informações do usuário temos que organizá-las racionalmente em um documento, na **Figura 1** temos um modelo.

Casos de uso

Como já mencionado, nossa aplicação é um catálogo das revistas ClubeDelphi e MSDN Magazine que será utilizado para futuras consultas, portanto podemos começar com a descrição do Caso de Uso que mostre como cadastrar um artigo (**Figura 2**).

Observe que no *Cenário Principal* descrevemos de forma específica as tarefas envolvidas para se cadastrar um artigo, todas enumeradas na seqüência que o cliente deseja. Não foram envolvidos termos técnicos porque como mencionado antes, um Caso de Uso sempre detalha a visão do usuário.

Também não são mencionadas exceções, isso porque o cenário principal representa a situação perfeita. A seção *Cenário Alternativo* é utilizada para realizar a descrição das exceções referentes cada passo do *Cenário Principal*.

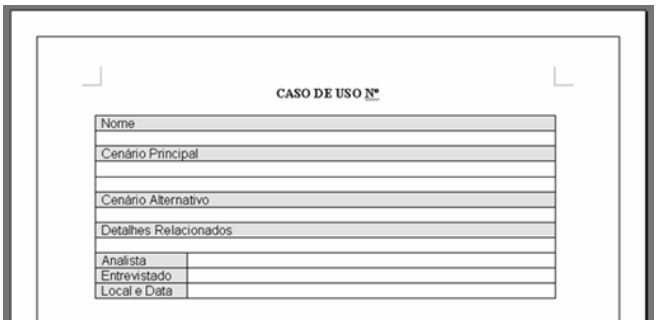
As exceções possuem uma numeração constituída da referência do passo a que se aplica, mais sua ordem dentro daquele passo. No passo 2 do *Cenário Principal* fazemos uma referência ao Caso de Uso *Cadastrar revista*, no entanto ele ainda não está desenvolvido, vamos por a mão na massa então. Na **Figura 3** temos o Caso de Uso pronto.

Vamos agora para a lógica central do negócio, a consulta dos artigos. Veja na **Figura 4** seu Caso de Uso (chamado *Consultar artigos*).

Baseado nas descrições, vamos agora, finalmente, modelar nossos Casos de Uso e para isso vamos utilizar a ferramenta ModelMaker.

ModelMaker

O ModelMaker é uma grande ferramenta para o desenvolvimento de projetos Orientados a Objetos, pois possui



CASO DE USO Nº

Nome
Cenário Principal
Cenário Alternativo
Detalhes Relacionados

Analista	
Entrevistado	
Local e Data	

Figura 1. Modelo de formulário para Caso de Uso

CASO DE USO Nº 1

Nome	
Cadastrar artigos	
Cenário Principal	
1 – Informa o nome do artigo 2 – Escolhe a partir de uma lista a revista (ver caso de uso Cadastrar revista) 3 – Informa o número da Edição 4 – Informa um resumo do artigo 5 – Informa palavras-chave	
Cenário Alternativo	
2.1 – Se a revista que se deseja não consta na lista, permitir seu cadastro	
Detalhes Relacionados	
Analista	Paulo
Entrevistado	Paulo
Local e Data	Jaboticabal, 26/04/2006

Figura 2. Caso de uso Cadastrar Artigos

CASO DE USO Nº 2

Nome	
Cadastrar revista	
Cenário Principal	
1 – Informa o nome da revista 2 – Informa email para contato 3 – Informa link da revista	
Cenário Alternativo	
Detalhes Relacionados	
Analista	Paulo
Entrevistado	Paulo
Local e Data	Jaboticabal, 26/04/2006

Figura 3. Caso de uso Cadastrar Revista

CASO DE USO Nº 3

Nome	
Consultar artigos	
Cenário Principal	
1 – Escolhe o tipo busca desejada a partir de uma descrição. Essa busca pode ser por nome do artigo, palavra-chave ou resumo do artigo 2 – Escolhe se aplica ou não um filtro por Revista 3 – Informa o texto a se buscar 4 – Visualiza o resultado da busca em uma lista 5 – Seleciona determinado artigo da lista 6 – Realiza manutenção no artigo selecionado, compreende-se por manutenção edição ou exclusão do artigo selecionado	
Cenário Alternativo	
3.1 – Se o texto informado estiver vazio a busca não é realizada e uma notificação é feita 5.1 – Não existindo resultado para busca, uma notificação é feita	
Detalhes Relacionados	
Analista	Paulo
Entrevistado	Paulo
Local e Data	Jaboticabal, 26/04/2006

Figura 4. Exemplo de Caso de Uso, chamado Consultar artigos

ClubeDelphi 76.indb 29 27.07.06 15:39:07

uma ótima integração com o Delphi e utiliza a UML para realização de análise e documentação (Digramas de Caso de Uso, Classes, Estado etc.).

Podemos realizar o download do instalador da última versão (no fechamento desta edição era a 8.20) no site *www.modelmakertools.com/modelmaker/index.html*, ou usar a versão que acompanha o Delphi 7. Depois de realizado download, sua instalação é simplificada com apenas uma ressalva, onde devemos instalar o ModelMaker com o IDE do Delphi fechado. Ao abrirmos o Delphi já notaremos um novo menu, o ModelMaker (**Figura 5**).

Com essa integração, conseguimos não apenas abrir o ModelMaker, mas enviar *units* que contenham nossas classes para visualizá-las através de um Diagrama de Classes, ainda com a possibilidade de alterá-las no diagrama e o código a ser atualizado. Podemos até fazer o contrário, alterar o código e ter o diagrama alterado. Ainda podemos criar nossas classes direto no ModelMaker, através de diagramas e ter o código-fonte gerado no Delphi.

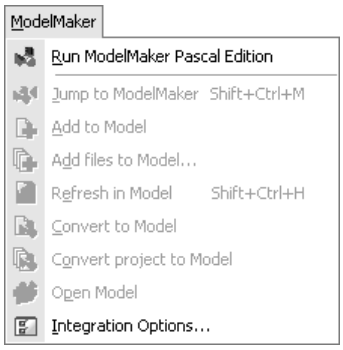


Figura 5. Menu ModelMaker dentro do IDE do Delphi

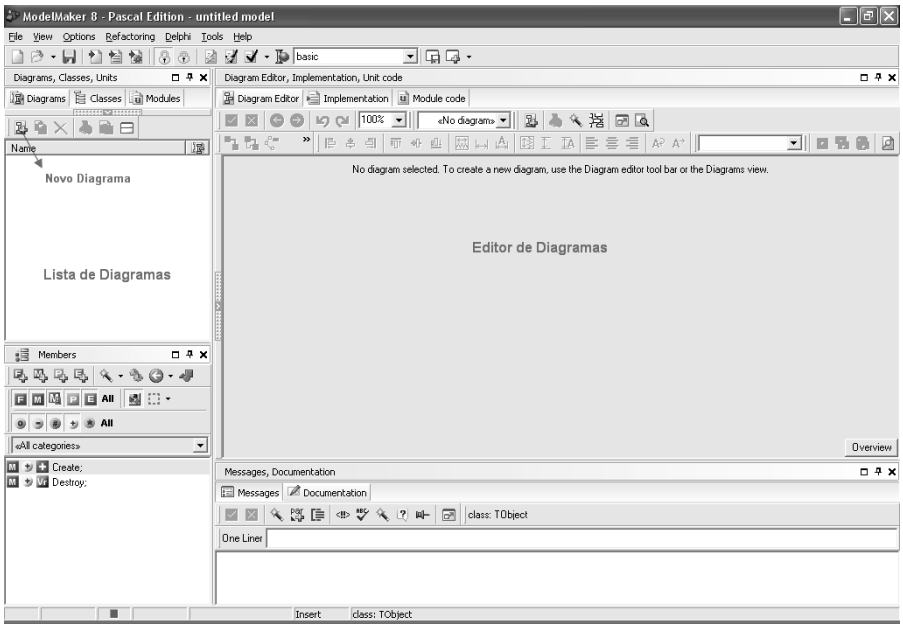


Figura 6. Tela inicial do ModelMaker

O trabalho em conjunto dessas grandes ferramentas pode agilizar o desenvolvimento Orientado a Objetos não apenas com sua geração automática de código, mas por permitir que um sistema seja disposto nos modelos da UML de forma fácil.

Tornando sua compreensão melhorada e em meu ponto de vista, esse é um fator fundamental no desenvolvimento, a compreensão da regra de negócio envolvida. Ao abrir o ModelMaker a tela da **Figura 6** é apresentada. Inseri alguns comentários que indicam o básico para iniciarmos nosso Diagrama de Caso de Uso.

Caso de Uso número 1

O Caso de Uso é um diagrama que exibe de forma superficial o que o usuário deseja que o sistema realize. Para implementá-lo, clique no botão *Add Diagram* para que a lista de diagramas seja exibida.

Na lista, escolha *Use Case Diagram*. Vamos renomeá-lo para "Cadastrar Artigos", para isso clique sobre seu nome e pressione F2. Observe que temos a barra de objetos para exibir os símbolos pertencentes a um diagrama de Caso de Uso, segundo a UML. Na **Figura 7** temos uma descrição desses símbolos que serão explicados durante a modelagem.

De posse da descrição do Diagrama de Caso de Uso número 1, vemos que o usuário interage com o sistema, isso significa que ele possui o papel de "Ator" em nosso Caso de Uso. Um ator pode ser qualquer pessoa, sistema ou equipamento que interaja com o sistema. Vamos inserir o Ator em nosso diagrama e renomeá-lo para "Usuario", clique sobre o botão *Actor* e clique em alguma área do editor de diagrama.

Posteriormente adicione um container para organização e o nomeie como "Cadastro de Artigos". Selecione o botão *Use Case* e o adicione no container. Um Caso de Uso representa uma atividade, uma macro-ação que um ator pode realizar, e para o Caso de Uso inserido anteriormente, vamos dar o nome de "Cadastrar artigos".

Voltemos agora à descrição do Caso de Uso número 1. No cenário alternativo é indicado que, a partir do cadastro de artigos, podemos cadastrar uma revista. Como vamos indicar isso? A UML oferece uma forma de exibir graficamente essa relação através do símbolo *Include*. Antes de utilizar o *Include* devemos inserir o Caso de Uso que pode ser utilizado no Caso de Uso Principal.

Adicione mais um caso dando o nome de "Cadastrar revista". Devemos agora informar a relação entre os Casos de Uso, para isso clique no botão *Include* e arraste do *Cadastrar revista* para *Cadastrar artigo*. Falta agora indicar qual caso que o ator realiza.

ClubeDelphi 76.indb 30 27.07.06 15:39:08

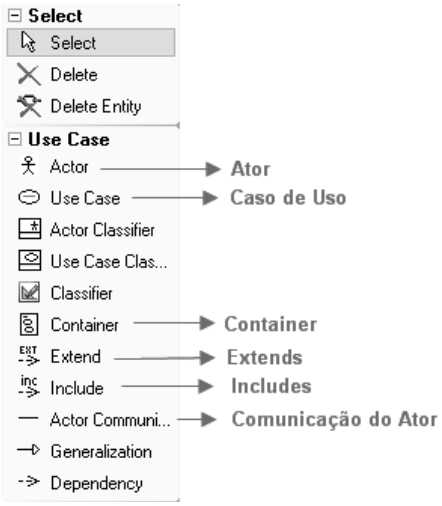


Figura 7. Barra de objetos para um Caso de Uso

Conforme visto na **Figura 7** temos o botão *Actor Communication*, é através dele que indicamos a relação Ator-Caso.

Clique sobre o botão e arraste a partir o ator *Usuario* até *Cadastrar artigo*. Ao término, nosso caso de uso deve ser semelhante a **Figura 8.**

Para construir os próximos, siga os mesmo passos descritos anteriormente, tomando por base as **Figuras 9 e 10.**

Conclusão

A modelagem de Diagramas de Caso de Uso oferece ao desenvolver um completo conhecimento do domínio do problema. Ao documentar cada ação do usuário entendemos suas necessidades, e esses documentos gerados, serão a base para o desenvolvimento do software em si servindo como guia de forma a atender satisfatoriamente os requisitos do usuário.

Não perca na próxima edição, nossa escolha da camada de persistência e como definir o Diagramas de Classes. ■

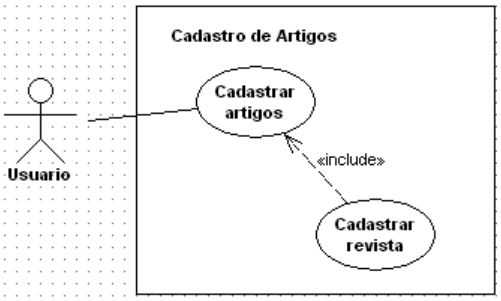


Figura 8. Caso de uso número 1 (Cadastro de Artigos)

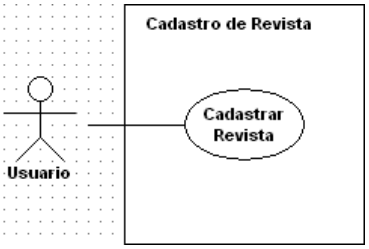


Figura 9. Caso de Uso número 2 (Cadastro de Revista)

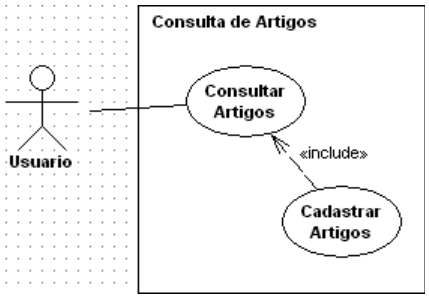


Figura 10. Caso de Uso número 3 (Consulta de Artigos)

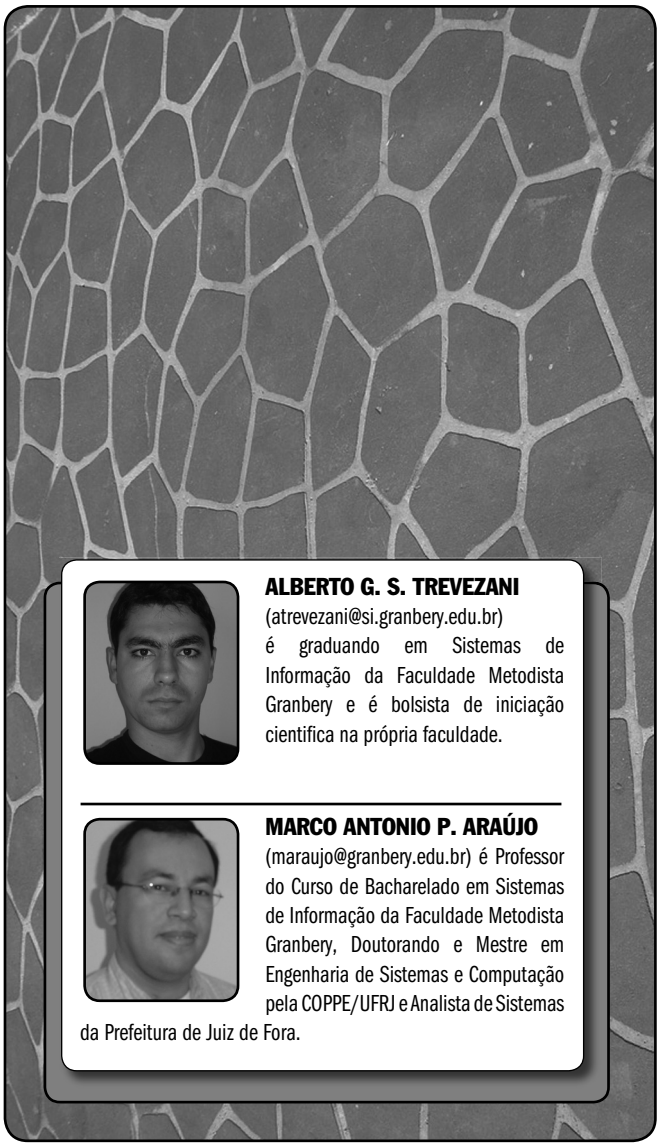
+ de 80.000 membros cadastrados
+ de 15.000 exemplos com fontes
+ de 900 apostilas
+ de 4.000 dicas
Fórum Delphi
Artigos
TOTALMENTE GRÁTIS
www.delphi.eti.br
Um dos maiores sites de apoio a desenvolvedores Delphi do Brasil!!!

ClubeDelphi 76.indb 31 27.07.06 15:39:08

ALBERTO G. SEVAROLLI TREVEZANI E MARCO ANTÔNIO P. ARAÚJO

Padrões de Projeto e POO

Integração com o Together no Delphi 2006



ALBERTO G. S. TREVEZANI
(atrevezani@si.granbery.edu.br)
é graduando em Sistemas de Informação da Faculdade Metodista Granbery e é bolsista de iniciação cientifica na própria faculdade.

MARCO ANTONIO P. ARAÚJO
(maraujo@granbery.edu.br) é Professor do Curso de Bacharelado em Sistemas de Informação da Faculdade Metodista Granbery, Doutorando e Mestre em Engenharia de Sistemas e Computação pela COPPE/UFRJ e Analista de Sistemas da Prefeitura de Juiz de Fora.

Com a difusão do desenvolvimento Orientado a Objetos e a necessidade de produzir software com mais qualidade, muitos padrões estão sendo incorporados ao desenvolvimento, com o objetivo de aumentar o reuso e a produtividade.

Os padrões mais conhecidos são denominados Padrões de Projeto (*Design Patterns*) ou Padrões GoF (*Gang of Four* - Gangue dos Quatro), que foram propostos em 1994 por *Erich Gamma, Richard Helm, Ralph Johnson* e *John Vlissides*, com a finalidade de descrever soluções para problemas recorrentes no desenvolvimento de software orientado a objetos.

Para facilitar sua utilização, alguns IDEs, como o Delphi, incorporaram esses padrões e oferecem mecanismos para adicionar os recursos no desenvolvimento de aplicações. Nesse sentido, este artigo apresenta o *Pattern Wizard*, uma ferramenta disponibilizada no Delphi 2006 que oferece vários padrões de projeto para facilitar o desenvolvimento de sistemas OO.

Padrões de Projeto

Padrões de projeto são soluções customizadas e reutilizáveis para problemas que são encontrados frequentemente no processo de desenvolvimento de aplicativos. Eles tratam da concepção e da interação entre objetos, definindo ao mesmo tempo um padrão de comunicação que é compartilhado por toda a equipe de desenvolvimento.

Os padrões de projetos são organizados em famílias de padrões: de criação, estruturais e comportamentais. Os padrões de criação são relacionados à criação de objetos, os estruturais tratam das associações entre classes e objetos e os comportamentais das interações e divisões de responsabilidades entre as classes ou objetos.

Um padrão de projeto é classificado também segundo o seu escopo: de classe ou de objeto. Nos padrões com escopo

ClubeDelphi 76.indb 32 27.07.06 15:39:09

de classe os relacionamentos que definem esse padrão são definidos através de herança e em tempo de compilação.

Nos padrões com escopo de objeto, o padrão é encontrado nos relacionamentos entre os objetos definidos em tempo de execução. Os diversos padrões de projeto estão divididos por categoria conforme a **Tabela 1.**

Por que utilizar padrões de projeto?

Um dos atrativos para a utilização dos padrões de projeto é que eles foram criados com base na experiência de outras pessoas, refinados com o tempo, testados e documentados. Além disso, as soluções produzidas utilizam boas práticas de programação Orientada a Objetos, explorando eficientemente conceitos como polimorfismo, herança, composição e encapsulamento, tornando um projeto flexível e reutilizável.

Com a utilização de soluções conhecidas, a comunicação da equipe de desenvolvimento é facilitada, permitindo que se trabalhe em um nível mais alto de abstração. A documentação e a compreensão do sistema também são facilitadas quando a solução é conhecida.

Dessa forma, utilizando os padrões de projeto é possível ter um controle maior da aplicação, aumentando a produtividade e tornando o custo de desenvolvimento menor.

Estudo de Caso com o Pattern Wizard

Com a maior integração do Delphi 2006 com o Together, foram disponibilizados recursos como definição de diagramas de casos de uso e de classes, e utilização de alguns padrões de projeto. O *Pattern Wizard* é esse utilitário, capaz de adicionar padrões de projeto a partir de um diagrama de classes. Além de oferecer alguns padrões de projeto, o *Pattern Wizard* oferece a possibilidade do desenvolvedor criar o seu próprio padrão.

Para ilustrar a utilização do *Pattern Wizard*, é apresentado o seguinte fragmento de diagrama de classes de um sistema de controle acadêmico (**Figura 1**), onde será demonstrada a utilização dos padrões de projeto *Singleton* e *Façade*.

Nesse estudo de caso o padrão *Façade* (fachada) será utilizado para simplificar o acesso às classes de negócio e o padrão *Singleton* não permitirá que a fachada (classe que representa um subsistema) seja instanciada mais de uma vez, garantido que apenas uma instância da mesma seja criada.

A **Tabela 2** demonstra as características principais do padrão de projeto *Singleton* e a **Tabela 3** demonstra as características do padrão de projeto *Façade*.

		Propósito		
		Criação	**Estrutural**	**Comportamental**
Escopo	**Classe**	Factory Method	Adapter (classe)	Interpreter Template Method
	Objeto	Abstract Factory Builder Prototype Singleton	Adapter (objeto) Bridge Composite Decorator Facade Flyweight Proxy	Chain of Responsibility Command Iterator Mediator Memento Observer State Strategy Visitor

Tabela 1. Divisão por categoria entre os padrões de projeto

Padrão Singleton	
Classificação	**Criação**
Propósito	Garantir que uma classe só tenha uma única instância e prover um ponto de acesso global a esta instância.
Motivação	Garantir que apenas um objeto exista independente do número de requisições para criá-lo.
Exemplos de aplicação	- Um único banco de dados; - Um único acesso ao arquivo de log.

Tabela 2. Características do padrão de projeto Singleton

Padrão Façade	
Classificação	**Estrutura**
Propósito	Possuir uma classe que representará um conjunto de classes (um subsistema), facilitando a interação com essas classes.
Motivação	Providenciar uma classe fachada (classe que representará um subsistema) que abstrai todo o subsistema tornando-o mais fácil de ser utilizado, pois existe apenas uma interface para cada subsistema.
Exemplos de aplicação	- Classe de um mecanismo de persistência que encapsule todo o acesso ao banco de dados.

Tabela 3. Características do padrão de projeto Façade

ClubeDelphi 76.indb 33 27.07.06 15:39:12

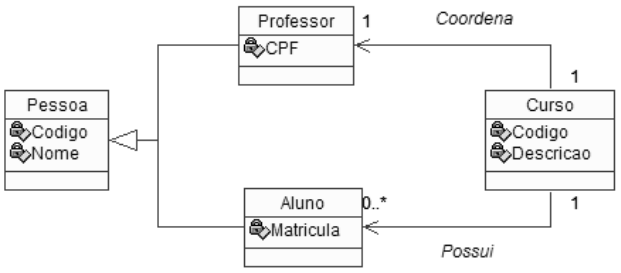


Figura 1. Diagrama de Classes

Com a estratégia definida já é possível iniciar o projeto no Delphi. Para isso basta acessar a opção do menu *File>New>VCL Forms Application - Delphi for Win32*. Nesse estudo de caso, o projeto será criado com o nome "PadroesDeProjeto.dpr" e a unit como "untPrincipal.pas".

Como o suporte à modelagem não é habilitado na criação do projeto, essa ativação deverá ser feita manualmente clicando na aba *Model View*. Será perguntado se deve ser adicionado ao projeto o suporte à modelagem, bastando responder que sim (**Figura 2**).

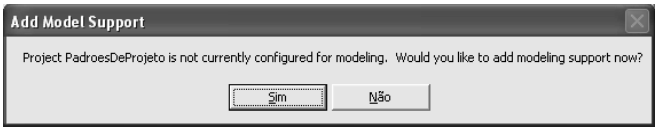


Figura 2. Confirmação para ativação do suporte a modelagem

Feito isso, o Delphi irá acionar a aba *Model View*, conforme **Figura 3.**

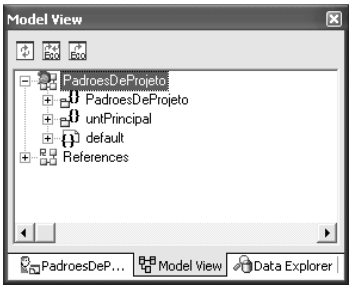


Figura 3. Model View habilitado

Ao ativar o suporte a modelagem, o Delphi adicionará automaticamente ao *Model View* os *Namespaces* já existentes no projeto atual (*PadroesDeProjeto*, neste exemplo). Nesse processo é criado ainda um *Namespace* para cada unit do projeto, como o caso da unit *untPrincipal*.

Antes da criação do diagrama de classes é necessário criar um *Namespace* para o diagrama. Para isso, basta selecionar o projeto no *Model View,* pressionar o botão direito do mouse e escolher a opção *Add>Namespace*. Feito isso, altere o nome padrão para "Diagramas".

Com o *Namespace* criado já é possível criar o diagrama de classes da aplicação bastando selecionar o *Namespace* criado, pressionar o botão direito do mouse e escolher a opção *Add>Class Diagram*, adicionando ao projeto um diagrama de classes.

Feitos esses passos, torna-se possível adicionar padrões de projeto à aplicação. Assim, basta clicar com o botão direito do mouse sobre o diagrama de classes e selecionar a opção *Create by Pattern* (**Figura 4**).

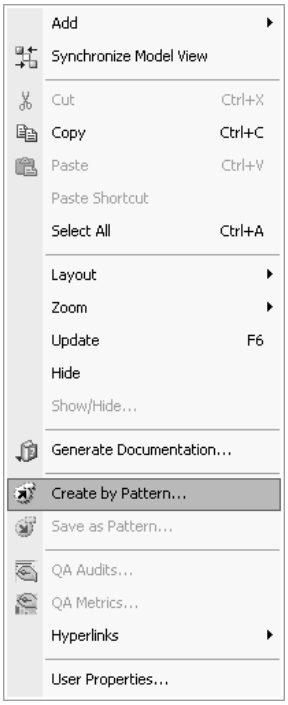


Figura 4. Opção para criação do diagrama de classes

Será apresentada a janela do *Pattern Wizard* (**Figura 5**) com os padrões suportados, separados por categoria.

O primeiro padrão a ser adicionado no diagrama de classe deverá ser o *Singleton*. Para adicioná-lo, basta selecionar

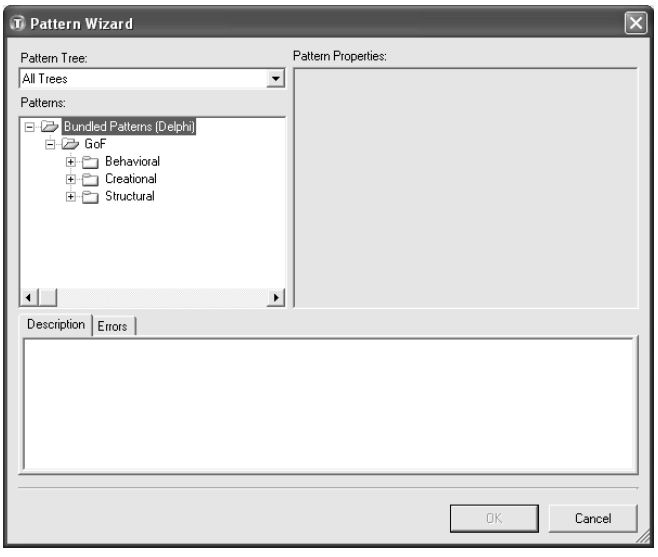


Figura 5. Janela de seleção de padrões do Pattern Wizard

ClubeDelphi 76.indb 34 27.07.06 15:39:13

o item *Creational* na árvore de padrões e selecioná-lo. A seguir, é exibida sua descrição contendo objetivo, classes participantes e as conseqüências da utilização do padrão.

Além disso, o *Pattern Wizard* oferece a opção de alterar os nomes das classes a serem geradas, através da opção *Class Singleton*. Para esse estudo de caso, o nome deverá ser alterado para "TSistema" (**Figura 6**).

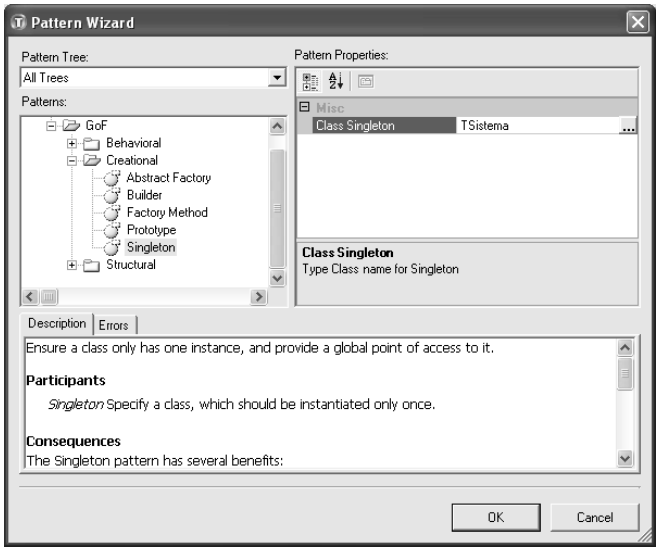


Figura 6. Opções do Pattern Wizard para o padrão Singleton

Feito isso, o Delphi irá gerar no diagrama de classes a classe *TSistema*, associada a uma elipse informativa indicando que essa classe representa o padrão de projeto *Singleton* (**Figura 7**).

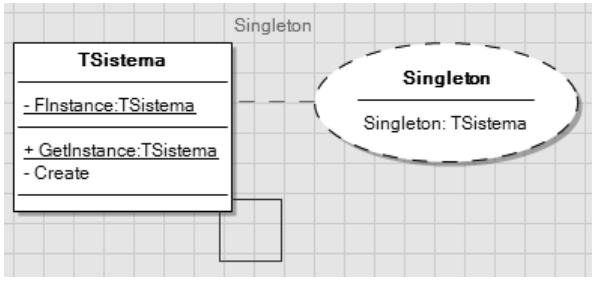


Figura 7. Diagrama de Classes com o padrão Singleton

> **Nota:** Caso o diagrama não seja atualizado, feche e abra-o novamente, ou tecle F6.

Como observado, a classe *TSistema* possui uma auto-associação necessária para retornar o único objeto, seguindo os seguintes passos:

- *1º Passo*: é solicitada uma instância do objeto *TSistema* através do método de classe *GetInstance*. Essa deve ser a única forma de obter o objeto dessa classe e, para isso, deve-se observar que o serviço construtor (*Create*) é privado, de forma a não permitir que seja chamado a partir de outra classe ou objeto;
- *2º Passo*: o método verifica se já existe uma instância desse objeto, verificando o conteúdo do atributo de classe *FInstance*;
- *3º Passo*: caso o atributo não contenha um objeto é criada uma instância desse através do método privado *Create*;
- *4º Passo*: o método *GetInstance* retorna a instância do objeto.

Com o padrão *Singleton* criado, o próximo passo é adicionar o padrão *Façade*. Para isso, basta repetir o mesmo processo, pressionando o botão direito do mouse e selecionando a opção *Create by Pattern*.

Para adicionar o padrão *Façade* basta selecionar o item *Structural* na árvore de padrões e selecioná-lo. A seguir, na opção *Class Façade*, deve-se selecionar a classe *TSistema* do diagrama de classes que, além de implementar o padrão *Singleton*, também será a fachada para as classes que comporão o subsistema.

Também é possível definir a primeira classe que irá compor o subsistema, que será a classe *TPessoa* (**Figura 8**).

O resultado obtido é apresentado na **Figura 9**.

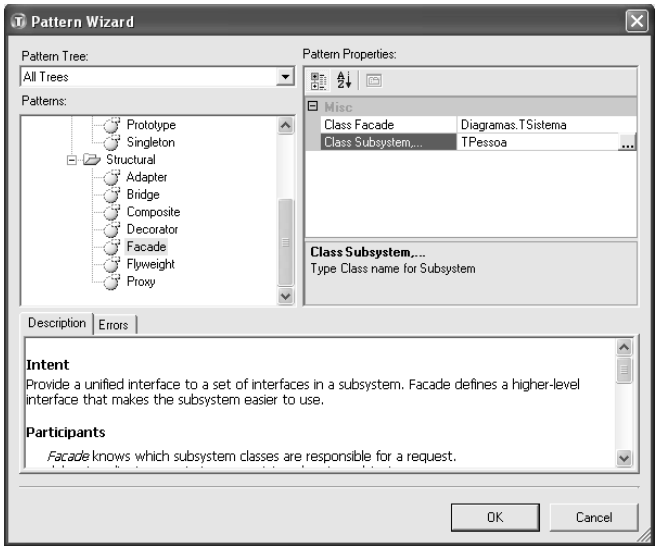


Figura 8. Opções do Pattern Wizard para o padrão Façade

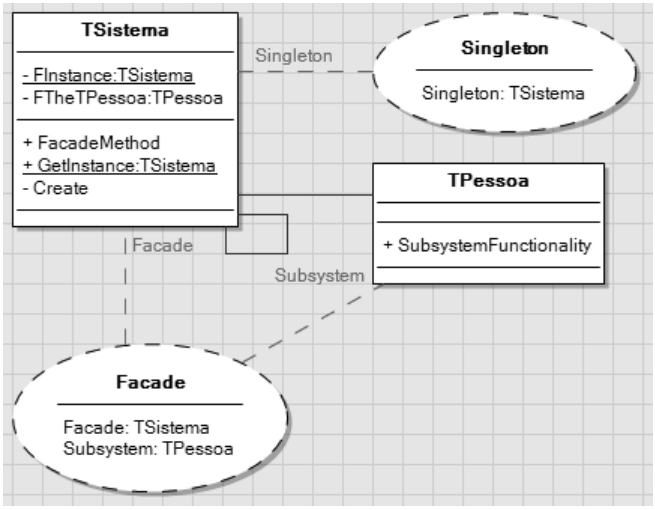


Figura 9. Diagrama de Classes com o os padrões Singleton e Façade

ClubeDelphi 76.indb 35 27.07.06 15:39:13

Observa-se nessa figura que a elipse criada pelo padrão *Façade* está ligada a duas classes, com os rótulos *Façade* (que indica a classe *TSistema* que implementa o padrão) e *Subsystem* (que indica a classe *TPessoa* que será acessada pela classe que implementa o padrão).

Ainda verifica-se uma associação entre as classes *TSistema* e *TPessoa*, de forma que a primeira possa referenciar a segunda. Com isso, o próximo passo é adicionar ao *Façade* as demais classes do estudo de caso. Para isso, basta clicar com o botão direito do mouse na elipse *Façade* e selecionar a opção *Add>Subsystem*. Essa opção serve para adicionar novas classes ao *Façade*. A seguir, deve-se alterar o nome da classe (adicione as classes: "TAluno", "TProfessor" e "TCurso").

Para adicionar atributos às classes, basta clicar com o botão direito do mouse sobre a classe e selecionar a opção *Add>Field*. Para adicionar serviços às classes, basta clicar com o botão direito do mouse sobre a classe e selecionar a opção *Add>Function*.

No modelo de classes proposto, as classes *TAluno* e *TProfessor* herdam da classe *TPessoa*. Para criar essa herança basta selecionar na *Tool Pallete* (**Figura 10**) o item *Generalization* e clicar na classe *TAluno* e depois na classe *TPessoa*.

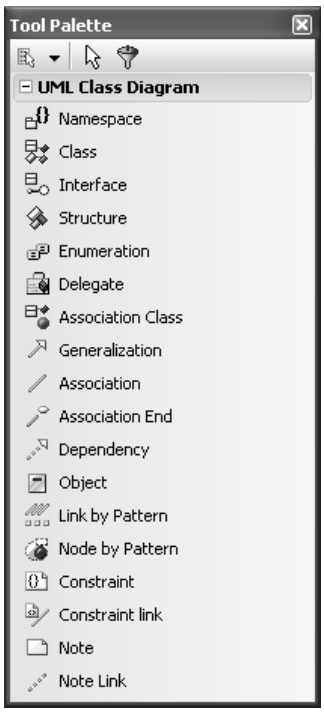


Figura 10. Tool Palette com a aba UML Class Diagram

O mesmo processo deve ser feito para a classe *TProfessor*. Para fazer as associações basta selecionar na *Tool Pallete* o item *Association* e clicar na classe *TAluno* (*Client*) e depois na classe *TCurso* (*Supplier*). Ao criar uma associação, é automaticamente inserida na classe cliente (*Client*) um atributo de ligação com a classe fornecedora (*Supplier*). Nesse exemplo, esse atributo foi renomeado para "Curso", através do *Object Inspector*, indicando que um aluno é matriculado em um curso.

As multiplicidades podem ser definidas no *Object Inspector* clicando na associação e alterando a propriedade *Supplier Cardinality* para definir a multiplicidade da classe fornecedora e alterando a propriedade *Client Cardinality* para definir a multiplicidade da classe cliente, conforme **Figura 11.**

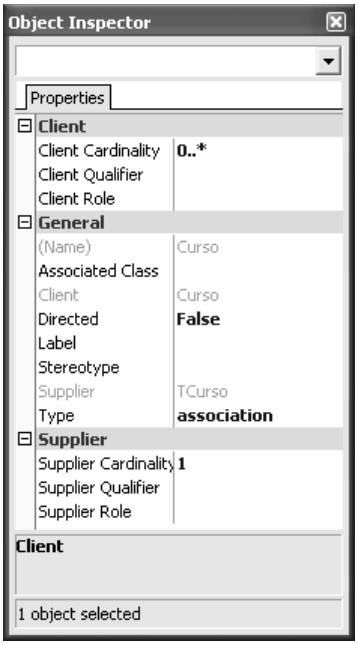


Figura 11. Object Inspector

O mesmo processo deve ser feito para a classe *TProfessor*. O resultado final é demonstrado na **Figura 12**.

Nessa figura é possível verificar o resultado final do diagrama de classes proposto pelo estudo de caso com a utilização dos padrões de projeto. Na classe *TSistema* foram criados automaticamente os atributos *FTheTAluno*, *FTheTCurso*, *FTheTPessoa* e *FTheProfessor* que são referências para objetos das classes referenciadas pelo padrão *Façade*.

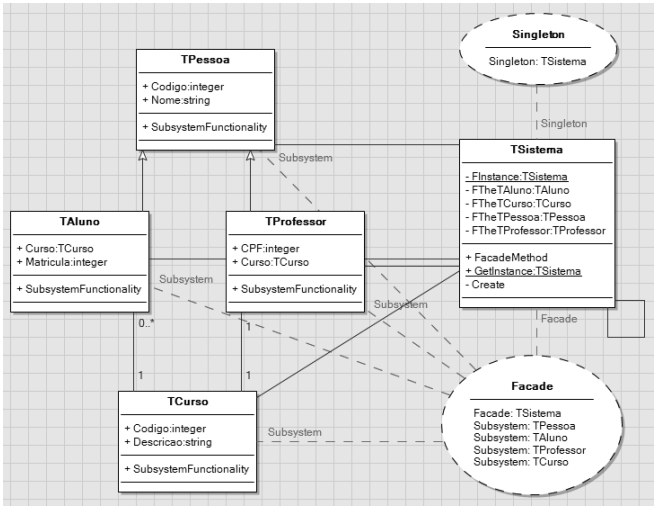


Figura 12. Diagrama de classes final com os padrões Singleton e Façade e as classes de negócio

ClubeDelphi 76.indb 36 27.07.06 15:39:14

Listagem 1. Código gerado pelo diagrama de classes

```
01: unit Diagramas;
02:
03: interface
04:
05: type
06:   TCurso = class;
07:   TProfessor = class;
08:   TAluno = class;
09:   TPessoa = class;
10:
11:   TSistema = class
12:   strict private
13:     constructor Create;
14:
15:   var
16:     FTheTPessoa: TPessoa;
17:     FTheTAluno: TAluno;
18:     FTheTProfessor: TProfessor;
19:     FTheTCurso: TCurso;
20:
21:   class var
22:     FInstance: TSistema;
23:
24:   public
25:     class function GetInstance: TSistema;
26:     procedure FacadeMethod;
27:   end;
28:
29:   TPessoa = class
30:   public
31:     procedure SubsystemFunctionality;
32:
33:   var
34:     Codigo: integer;
35:     Nome: string;
36:   end;
37:
38:   TAluno = class(TPessoa)
39:   public
40:     procedure SubsystemFunctionality;
41:
42:   var
43:     Matricula:integer;
44:     //<clientCardinality>0..*</clientCardinality>
45:     //  <supplierCardinality>1
        //   </supplierCardinality>
46:     Curso: TCurso;
47:   end;
48:
49:   TProfessor = class(TPessoa)
50:   public
51:     procedure SubsystemFunctionality;
52:
53:   var
54:     CPF:integer;
55:     //<clientCardinality>1</clientCardinality>
56:     //  <supplierCardinality>1
        //  </supplierCardinality>
57:     Curso: TCurso;
58:   end;
59:
60:   TCurso = class
61:   public
62:     procedure SubsystemFunctionality;
63:
64:   var
65:     Codigo: integer;
66:     Descricao: string;
67:   end;
68:
69: implementation
70:
71: constructor TSistema.Create;
72: begin
73:   inherited;
74: end;
75:
76: class function TSistema.GetInstance: TSistema;
77: begin
78:   If FInstance = nil Then
79:   begin
80:     FInstance := TSistema.Create();
81:   end;
82:   Result := FInstance;
83: end;
84:
85: procedure TSistema.FacadeMethod;
86: begin
87:   // multiple subsystems could be involved
88:   // to provide some functionality
89: end;
90:
91: procedure TPessoa.SubsystemFunctionality;
92: begin
93: end;
94:
95: procedure TAluno.SubsystemFunctionality;
96: begin
97: end;
98:
99: procedure TProfessor.SubsystemFunctionality;
100: begin
101: end;
102:
103: procedure TCurso.SubsystemFunctionality;
104: begin
105: end;
106:
107: end.
```

Além disso, também em função desse padrão de projeto, foi criado um método *FacadeMethod* na classe *TSistema* e, em cada classe do subsistema, foi criado automaticamente um método *SubsystemFunctionality* exemplificando a utilização de métodos nessas classes.

O código fonte é gerado automaticamente e pode ser visualizado através de um duplo clique em qualquer elemento do diagrama, ou selecionar uma das classes do diagrama e, com o botão direito do mouse, selecionar a opção *Go To Definition*. Veja na **Listagem 1** o código-fonte.

Em relação ao padrão *Singleton*, pode-se observar a definição do serviço construtor privado nas linhas 12 e 13. Nas linhas 21 e 22 é definido o atributo de classe *FInstance*, responsável por armazenar o objeto único da classe *TSistema*. A implementação do serviço de retorno desse objeto único é descrita nas linhas 76 a 83.

Quanto ao padrão *Façade*, pode-se observar nas linhas 16 a 19 a definição dos atributos dos objetos das classes referenciadas pelo padrão. As linhas 44, 45, 55 e 56 apresentam comentários referentes às multiplicidades definidas no diagrama de classes. Nas linhas 85 a 105 são definidos os métodos de exemplo das classes que compõem o padrão *Façade*.

Finalmente, as linhas 46 e 57 apresentam os atributos da associação das classes *TAluno* e *TProfessor* com a classe *TCurso*, respectivamente.

Conclusão

Conforme observado, a utilização de padrões de projeto com o Delphi 2006 não é complicada, tornando possível adicioná-los de forma rápida e simples. O objetivo deste artigo foi mostrar as características do *Pattern Wizard* e apresentar alguns conceitos de padrões de projeto.

Para maior entendimento desse processo foi desenvolvido um estudo de caso, onde se procurou utilizar alguns padrões de projeto visando mostrar sua integração com o sistema. ■

Links

Site do Together

www.borland.com/br/products/together

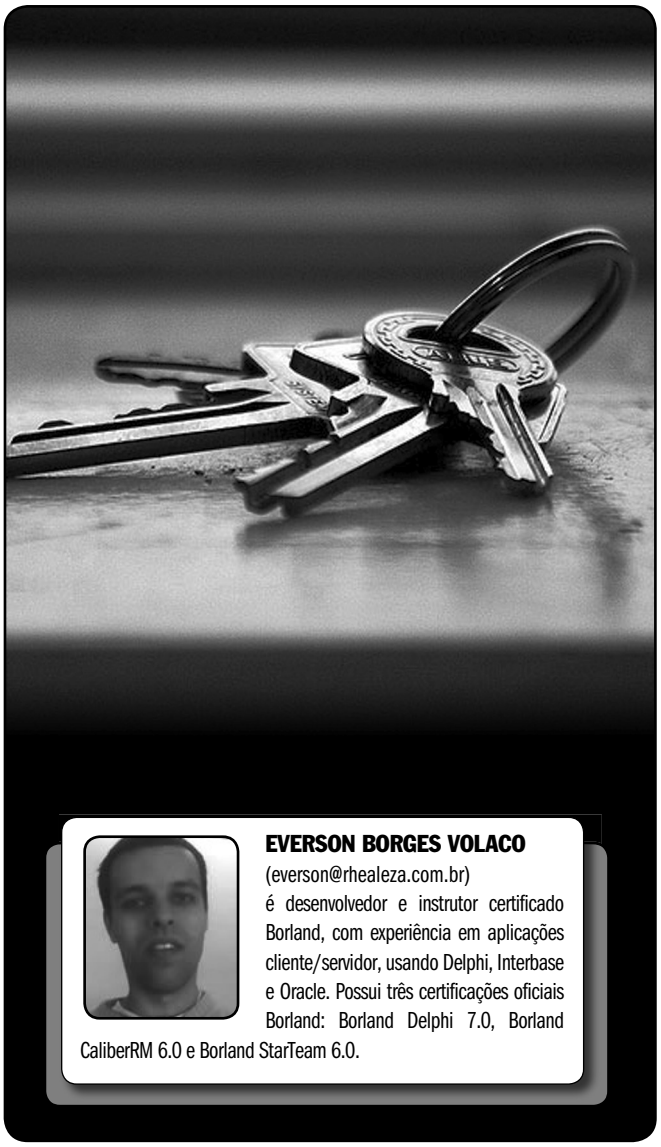
Portal OODesign - Biblioteca de Padrões de Projeto

www.oodesign.com.br/patterns

ClubeDelphi 76.indb 37 27.07.06 15:39:14

Controle de Acesso

Realizando o controle de acesso de usuários e grupos da aplicação através de menu dinâmico



EVERSON BORGES VOLACO
(everson@rhealeza.com.br)
é desenvolvedor e instrutor certificado Borland, com experiência em aplicações cliente/servidor, usando Delphi, Interbase e Oracle. Possui três certificações oficiais Borland: Borland Delphi 7.0, Borland CaliberRM 6.0 e Borland StarTeam 6.0.

Neste artigo veremos como realizar o controle de acesso em uma aplicação Delphi, utilizando menu dinâmico. A idéia é definir os privilégios de acesso dos usuários e grupos do sistema no banco de dados e habilitar as opções do menu principal (*MainMenu*) e dos botões das telas de cadastro, de acordo com esses privilégios.

> **Nota:** Para o exemplo deste artigo utilizei o Delphi 7 com o InterBase 7.5.1, porém você pode utilizar qualquer outra versão do Delphi e do InterBase ou Firebird.

Exemplo

Para o exemplo deste artigo criaremos uma aplicação VCL (Win32) contendo inúmeras opções de menu e telas de cadastro e movimentação. Para não sair do foco do assunto deste artigo, controle de acesso, não implementaremos todas as telas desse sistema, porém as funcionalidades que serão necessárias para fazer o controle de acesso dos usuários e grupos, serão mostradas.

Serão utilizados os privilégios de *Consulta*, *Inserção*, *Alteração* e *Exclusão*, os quais poderão ser definidos tanto para um grupo quanto para um usuário do sistema. Um usuário poderá pertencer a mais de um grupo, onde, caso o mesmo possua privilégios de acesso diferentes em cada grupo, os privilégios serão concatenados.

Por exemplo, se definirmos que o usuário *Everson* pertence aos grupos *Desenvolvedores* e *Analistas* e no grupo *Desenvolvedores* o mesmo tenha apenas permissão de *Inserção* na tela de clientes, porém, no grupo *Analistas* tenha permissões de *Alteração* e *Exclusão*, então o *Everson* será capaz de *Inserir*, *Alterar* e *Excluir* os clientes.

O controle de acesso poderá ser definido ainda ao nível

ClubeDelphi 76.indb 38 27.07.06 15:39:15

de usuário sem a necessidade de vinculá-lo a um ou mais grupos. No entanto, caso existam permissões de acesso definidas diretamente no usuário e esse pertença a um grupo que também possui permissões definidas, as mesmas serão concatenadas da mesma forma que ocorre entre os grupos.

> **Nota:** Por questões de espaço, mostraremos neste artigo apenas os códigos relacionados ao controle de acesso da aplicação. Você pode ter acesso ao código-fonte completo do exemplo através do link para download.

Definindo o banco de dados

Começaremos o exemplo, definindo as tabelas que serão responsáveis em armazenar as informações referentes ao controle de acesso do sistema. Abra a sua ferramenta de gerenciamento de banco de dados InterBase ou Firebird, como por exemplo o IBConsole ou IBExpert e crie um banco de dados com o nome de "ACESSO.IB". Criaremos seis tabelas no banco de dados, cuja descrição das mesmas consta na **Tabela 1.**

Nome da Tabela	Descrição
MENUS	Tabela contendo todas as opções disponíveis no menu principal da aplicação.
GRUPOS	Tabela contendo todos os grupos cadastrados no sistema.
USUARIOS	Tabela com todos os usuários do sistema.
GRUPOS_USUARIOS	Tabela de vínculo entre os usuários e grupos do sistema.
PERMISSOES_GRUPOS	Tabela contendo as permissões de acesso de cada item de menu e tela para os grupos do sistema.
PERMISSOES_USUARIOS	Tabela contendo as permissões de acesso de cada item de menu e tela para os usuários do sistema.

Tabela 1. Descrição das tabelas do banco

Na **Listagem 1** temos as instruções DDL para a criação das tabelas.

Listagem 1. Código para criação das tabelas

```
CREATE TABLE MENUS (
  MEN_CODIGO INTEGER NOT NULL,
  MEN_NOME VARCHAR(25) NOT NULL,
  MEN_CODIGOPAI INTEGER,
  MEN_EXISTE BOOLEAN NOT NULL,
  CONSTRAINT PK_MENUS PRIMARY KEY (MEN_CODIGO),
  CONSTRAINT FK_MENUS FOREIGN KEY (MEN_CODIGOPAI)
  REFERENCES MENUS(MEN_CODIGO)
  ON UPDATE CASCADE ON DELETE CASCADE);

CREATE TABLE GRUPOS (
  GRU_CODIGO INTEGER NOT NULL,
  GRU_DESCRICAO VARCHAR(30) NOT NULL,
  CONSTRAINT PK_GRUPOS PRIMARY KEY (GRU_CODIGO));

CREATE TABLE USUARIOS (
  USU_CODIGO INTEGER NOT NULL,
  USU_NOMECOMPLETO VARCHAR(70) NOT NULL,
  USU_LOGIN VARCHAR(15) NOT NULL,
  USU_SENHA VARCHAR(15) NOT NULL,
  USU_ADMIN BOOLEAN DEFAULT FALSE,
  CONSTRAINT PK_USUARIOS PRIMARY KEY (USU_CODIGO));

CREATE TABLE GRUPOS_USUARIOS (
  GRU_CODIGO INTEGER NOT NULL,
  USU_CODIGO INTEGER NOT NULL,
  CONSTRAINT PK_GRUPOS_USUARIOS
  PRIMARY KEY (GRU_CODIGO, USU_CODIGO),
  CONSTRAINT FK_GU_GRUPOS FOREIGN KEY (GRU_CODIGO)
  REFERENCES GRUPOS (GRU_CODIGO)
  ON UPDATE CASCADE ON DELETE CASCADE,
  CONSTRAINT FK_GU_USUARIOS FOREIGN KEY (USU_CODIGO)
  REFERENCES USUARIOS (USU_CODIGO)
  ON UPDATE CASCADE ON DELETE CASCADE);

CREATE TABLE PERMISSOES_GRUPOS (
  GRU_CODIGO INTEGER NOT NULL,
  MEN_CODIGO INTEGER NOT NULL,
  PG_NOVO BOOLEAN DEFAULT FALSE,
  PG_ALTERAR BOOLEAN DEFAULT FALSE,
  PG_APAGAR BOOLEAN DEFAULT FALSE,
  PG_CONSULTAR BOOLEAN DEFAULT FALSE,
  CONSTRAINT PK_PERMISSOES_GRUPOS
  PRIMARY KEY (GRU_CODIGO, MEN_CODIGO),
  CONSTRAINT FK_PG_USUARIOS FOREIGN KEY (GRU_CODIGO)
  REFERENCES GRUPOS (GRU_CODIGO)
  ON UPDATE CASCADE ON DELETE CASCADE,
  CONSTRAINT FK_PG_MENUS FOREIGN KEY (MEN_CODIGO)
  REFERENCES MENUS (MEN_CODIGO)
  ON UPDATE CASCADE ON DELETE CASCADE);

CREATE TABLE PERMISSOES_USUARIOS (
  USU_CODIGO INTEGER NOT NULL,
  MEN_CODIGO INTEGER NOT NULL,
  PU_NOVO BOOLEAN DEFAULT FALSE,
  PU_ALTERAR BOOLEAN DEFAULT FALSE,
  PU_APAGAR BOOLEAN DEFAULT FALSE,
  PU_CONSULTAR BOOLEAN DEFAULT FALSE,
  CONSTRAINT PK_PERMISSOES_USUARIOS
  PRIMARY KEY (USU_CODIGO, MEN_CODIGO),
  CONSTRAINT FK_PU_USUARIOS FOREIGN KEY (USU_CODIGO)
  REFERENCES USUARIOS (USU_CODIGO)
  ON UPDATE CASCADE ON DELETE CASCADE,
  CONSTRAINT FK_PU_MENUS FOREIGN KEY (MEN_CODIGO)
  REFERENCES MENUS (MEN_CODIGO)
  ON UPDATE CASCADE ON DELETE CASCADE);
```

> **Nota:** O Firebird não suporta campos do tipo *Boolean*. Você pode contornar, criando um campo *Char* de tamanho 1. Não esqueça de fazer as respectivas alterações no código do artigo

Para as tabelas *Grupos* e *Usuarios* vamos definir *Generators* e *Triggers* para realizar o auto incremento do valor da chave primária dessas tabelas. Na **Listagem 2** temos as instruções SQL para criação desses objetos no banco de dados.

Listagem 2. Generators e Triggers das tabelas

```
CREATE GENERATOR GEN_GRUPOS;

CREATE GENERATOR GEN_USUARIOS;

CREATE TRIGGER TRG_INC_GRUPOS
FOR GRUPOS BEFORE INSERT POSITION 0
AS
BEGIN
  IF (NEW.GRU_CODIGO IS NULL) THEN
    NEW.GRU_CODIGO = GEN_ID(GEN_GRUPOS, 1);
END

CREATE TRIGGER TRG_INC_USUARIOS
FOR USUARIOS BEFORE INSERT POSITION 0
AS
BEGIN
  IF (NEW.USU_CODIGO IS NULL) THEN
    NEW.USU_CODIGO = GEN_ID(GEN_USUARIOS, 1);
END
```


ClubeDelphi 76.indb 39 27.07.06 15:39:17

Para finalizar a criação dos objetos do banco de dados, vamos criar duas *Stored Procedures* que serão responsáveis em verificar os direitos de acesso do usuário logado no sistema. Na **Listagem 3** temos o código para criação da *Stored Procedure* PROC_VERIFICA_ACESSO_MENU, que receberá o código do usuário logado e o código do menu corrente como parâmetro de entrada e retornará o direito de acesso do usuário sobre o menu em questão.

Listagem 3. Criação da SP para retornar o direito de acesso do usuário

```
CREATE PROCEDURE PROC_VERIFICA_ACESSO_MENU (
  CODUSUARIO INTEGER, CODMENU INTEGER)
RETURNS (CONSULTAR CHAR(1))
AS
DECLARE VARIABLE GU_CONSULTAR BOOLEAN;
BEGIN
  CONSULTAR = 'N';
  FOR SELECT PU_CONSULTAR FROM PERMISSOES_USUARIOS
    WHERE USU_CODIGO = :CODUSUARIO AND
    MEN_CODIGO = :CODMENU
  UNION
  SELECT PG_CONSULTAR FROM PERMISSOES_GRUPOS
    WHERE GRU_CODIGO IN (
    SELECT GRU_CODIGO FROM GRUPOS_USUARIOS
    WHERE USU_CODIGO = :CODUSUARIO) AND
    MEN_CODIGO = :CODMENU
  INTO :GU_CONSULTAR
  DO
  BEGIN
    IF (:GU_CONSULTAR = TRUE) THEN
    BEGIN
      CONSULTAR = 'S';
      EXIT;
    END
  END
END
```

Para verificar os direitos de acesso do usuário a uma tela do sistema utilizaremos também uma *Stored Procedure*. A PROC_VERIFICA_DIREITOS_ACESSO descrita na **Listagem 4**, que retornará os direitos de inclusão, alteração e exclusão do usuário sobre a tela verificada.

Listagem 4. Procedure que retorna os direitos do usuário

```
CREATE PROCEDURE PROC_VERIFICA_DIREITOS_ACESSO (
  CODUSUARIO INTEGER, CODMENU INTEGER)
RETURNS (NOVO CHAR(1), ALTERAR CHAR(1),
  APAGAR CHAR(1))
AS
DECLARE VARIABLE ADM BOOLEAN;
DECLARE VARIABLE GU_NOVO BOOLEAN;
DECLARE VARIABLE GU_ALTERAR BOOLEAN;
DECLARE VARIABLE GU_APAGAR BOOLEAN;
BEGIN
  NOVO = 'N'; ALTERAR = 'N'; APAGAR = 'N';
  SELECT USU_ADMIN FROM USUARIOS
    WHERE USU_CODIGO = :CODUSUARIO
  INTO :ADM;
  IF (:ADM = TRUE) THEN
  BEGIN
    NOVO = 'S';
    ALTERAR = 'S';
    APAGAR = 'S';
    EXIT;
  END

  FOR SELECT PU_NOVO, PU_ALTERAR, PU_APAGAR
    FROM PERMISSOES_USUARIOS
    WHERE USU_CODIGO = :CODUSUARIO AND
    MEN_CODIGO = :CODMENU
  UNION
  SELECT PG_NOVO, PG_ALTERAR, PG_APAGAR
    FROM PERMISSOES_GRUPOS
    WHERE GRU_CODIGO IN (SELECT GRU_CODIGO
    FROM GRUPOS_USUARIOS
    WHERE USU_CODIGO = :CODUSUARIO) AND
    MEN_CODIGO = :CODMENU
  INTO :GU_NOVO, :GU_ALTERAR, :GU_APAGAR
  DO
  BEGIN
    IF (:GU_NOVO = TRUE) THEN
      NOVO = 'S';
    IF (:GU_ALTERAR = TRUE) THEN
      ALTERAR = 'S';
    IF (:GU_APAGAR = TRUE) THEN
      APAGAR = 'S';
  END
END
```

Construindo a aplicação Delphi

Inicie uma nova aplicação Delphi (*File>New>Application*) e altere o nome do formulário para "FrmPrincipal". Salve sua *unit* como "untFrmPrincipal.pas". Para o arquivo de projeto salve-o como "ControleAcesso.dpr".

Crie um Data Module (*File>New>Data Module*), altere seu nome para "DMPrincipal" e salve sua *unit* como "untDMPrincipal.pas". Adicione alguns componentes não-visuais de acesso a dados da paleta *dbExpress* (**Figura 1**).

Altere o nome dos componentes como mostrado na figura e configure o *SQLConnection* para acessar o banco *ACESSO.IB*. Selecione os quatro componentes do Data Module (com exceção do *conAcesso*) e aponte a propriedade *SQLConnection* para *conAcesso*.

No *spAcessoMenu*, altere a propriedade *StoredProcName* para PROC_VERIFICA_ACESSO_MENU. Para o *SPDireitoAcesso* aponte a propriedade *StoredProcName* para PROC_VERIFICA_DIREITOS_ACESSO.

> **Nota:** Manteremos a propriedade *CommandText* dos componentes *datasetAux* e *datasetAux2* vazia, pois utilizaremos os mesmos por diversas funções do sistema, que enviarão instruções SQL dinâmicas.

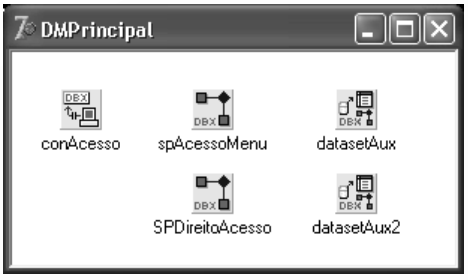


Figura 1. Componentes para acesso aos objetos do banco de dados

ClubeDelphi 76.indb 40 27.07.06 15:39:17

Voltando ao *FrmPrincipal* adicione um *MainMenu*, altere seu nome para "mmPrincipal" e através de um duplo clique inclua os seguintes itens de menu:

- Cadastros
  - Clientes
  - Fornecedores
  - Auxiliares
    - Cidades
    - Estados
    - Países
- Movimentações
  - Notas Fiscais
    - Entrada
    - Saída
  - Ordem de Serviço
- Segurança
  - Grupos
  - Usuários
  - Permissões de Grupo
  - Permissões de Usuário
- Outros
  - Calculadora
  - MS Excel
  - MS Word
  - Caixa Sobre

Utilizaremos a tabela MENUS do banco de dados para armazenar todos os itens de menu cadastrados no *MainMenu*. Esse cadastramento ocorrerá no momento em que o sistema for executado. A idéia é tornar esse cadastro o mais dinâmico possível, isso é, caso seja incluído um novo item o mesmo será cadastrado automaticamente na próxima vez que o sistema for executado.

O mesmo ocorrerá se um item for excluído do menu, nesse caso o item será removido da tabela na próxima execução do sistema. Antes de implementarmos essa funcionalidade altere a ordem de criação do *DMPrincipal* para que o mesmo seja criado antes do *FrmPrincipal*.

Para isso configure a seção *Auto-create forms* disponível na janela *Project Options>Forms*. No *FrmPrincipal* digite no evento *OnCreate* o código da **Listagem 5**.

No código da listagem anterior, utilizamos o *datasetAux* para atualizar a coluna MEN_EXISTE da tabela MENUS com o valor *False*. Na seqüência fazemos a chamada ao método *verificaOpcoesMenu*, que será detalhado mais adiante e é responsável em atualizar a tabela MENUS de acordo com os itens de menu disponíveis no *mmPrincipal*.

No final do evento realizamos a exclusão dos registros da tabela MENUS, onde a coluna MEN_EXISTE continua com o valor *False*. Veja na **Listagem 6** a implementação do método *verificaOpcoesMenu*.

Listagem 5. Código do evento OnCreate do FrmPrincipal

```
procedure TFrmPrincipal.FormCreate(Sender: TObject);
var
  i: Integer;
begin
  { Declare untDMPrincipal no uses }
  with DMPrincipal.datasetAux do
  begin
    Close;
    CommandText :=
      'UPDATE MENUS SET MEN_EXISTE = False';
    ExecSQL(True);
  end;
  for i := 0 to Pred(mmPrincipal.Items.Count) do
    verificaOpcoesMenu(mmPrincipal.Items[i]);
  with DMPrincipal.datasetAux do
  begin
    Close;
    CommandText :=
      'DELETE FROM MENUS WHERE MEN_EXISTE = False';
    ExecSQL(True);
  end;
end;
```

Listagem 6. Método VerificaOpcoesMenu

```
procedure TFrmPrincipal.verificaOpcoesMenu(
  const menu: TMenuItem);
var
  i, seq, codMenu, codMenuPai: Integer;
begin
  with DMPrincipal.datasetAux do
  begin
    Close;
    CommandText :=
      'SELECT MEN_CODIGO FROM MENUS WHERE MEN_NOME= '+
      QuotedStr(Trim(RetirarCaracterAtalho(
      menu.Caption)));
    Open;
    if IsEmpty then
    begin
      seq := retornaProximoCodigoMenu;
      if Assigned(menu.Parent) and
        (menu.Parent.Caption <> '') then
      begin
        codMenuPai := retornarCodMenu(Trim(
          RetirarCaracterAtalho(menu.Parent.Caption)));
        Close;
        CommandText :=
          'INSERT INTO MENUS (MEN_CODIGO, MEN_NOME, '+
          'MEN_CODIGOPAI, MEN_EXISTE) VALUES (' +
          IntToStr(seq) + ', ' +
          QuotedStr(Trim(RetirarCaracterAtalho(
          menu.Caption))) + ', ' + IntToStr(
          codMenuPai) + ', True)';
      end
      else
      begin
        Close;
        CommandText :=
         'INSERT INTO MENUS (MEN_CODIGO, MEN_NOME, '+
         'MEN_EXISTE) VALUES (' + IntToStr(seq) + ', '+
         QuotedStr(Trim(RetirarCaracterAtalho(
         menu.Caption))) + ', True)';
      end;
      ExecSQL(True);
    end
    else
    begin
      codMenu := Fields[0].AsInteger;
      Close;
      CommandText :=
        'UPDATE MENUS SET MEN_EXISTE = True '+
        'WHERE MEN_CODIGO = ' + IntToStr(codMenu);
      ExecSQL(True);
    end;
  end;
  for i := 0 to Pred(menu.Count) do
    verificaOpcoesMenu(menu.Items[i]);
end;
```

O método *verificaOpcoesMenu* recebe como parâmetro de entrada o item de menu corrente e realiza o cadastro do mesmo dentro da tabela MENUS. Como o item de menu em questão pode conter submenus, fazemos a chamada recursiva ao método.

ClubeDelphi 76.indb 41 27.07.06 15:39:18

Esse método utiliza ainda três métodos auxiliares para realizar as operações na tabela MENUS. Na **Listagem 7** temos essa implementação.

Listagem 7. Implementação dos métodos auxiliares

```
{ Utilizado para remover o "&" colocado
  automaticamente pelo Delphi em cada item
  de menu do MainMenu. }

function TFrmPrincipal.RetirarCaracterAtalho(
  const texto: string): string;
var
  i, tamanho : Integer;
begin
  tamanho := Length(texto);
  for i := 1 to tamanho do
    if texto[i] <> '&' then
      Result := Result + texto[i];
end;

{ Utilizado para trazer próximo código válido para
  inserção de um novo registro na tabela MENUS. }

function TFrmPrincipal.
  retornaProximoCodigoMenu: integer;
begin
  with DMPrincipal.datasetAux2 do
  begin
    Close;
    CommandText :=
      'SELECT MAX(MEN_CODIGO) AS MAIOR FROM MENUS';
    Open;
    Result := Fields[0].AsInteger + 1;
  end;
end;

{ Utilizado para retornar o código do menu passado
  como parâmetro de entrada. }

function TFrmPrincipal.retornarCodMenu(
  const textoMenu: string): Integer;
begin
  with DMPrincipal.datasetAux do
  begin
    Close;
    CommandText :=
      'SELECT MEN_CODIGO FROM MENUS WHERE MEN_NOME= '+
      QuotedStr(Trim(textoMenu));
    Open;
    if not IsEmpty then
      Result := Fields[0].AsInteger
    else
      Result := -1;
  end;
end;
```

Se rodarmos nossa aplicação, veremos que todos os itens de menu serão cadastrados automaticamente dentro da tabela MENUS. Crie formulários de cadastro simples para inclusão dos Grupos e Usuários do sistema. Esses cadastros serão utilizados apenas para a inserção, alteração e exclusão dos grupos e usuários que terão os direitos de acesso definidos.

> **Nota:** Você pode inserir alguns grupos e usuários diretamente no banco de dados. No exemplo disponível para download implementei tais telas de cadastro, onde, inclusive na tela de cadastro de usuários mostra como fazer a vinculação do mesmo a um ou mais grupos cadastrados armazenando a informação na tabela GRUPOS_USUARIOS.

Veja um layout de exemplo dos cadastros na **Figura 2.**

O próximo passo é definir os direitos de acesso de cada grupo e/ou usuário através das tabelas PERMISSOES_GRUPOS e PERMISSOES_USUARIOS. Da mesma maneira você pode criar uma tela para cadastro dessas permissões de acesso, ou, se preferir inseri-las diretamente no banco de dados. Veja na **Figura 3** um exemplo de tela para cadastro dos direitos de acesso para os grupos do sistema.

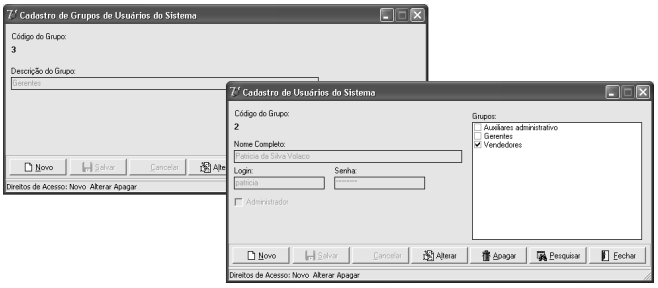


Figura 2. Cadastros de Grupos e Usuários do sistema

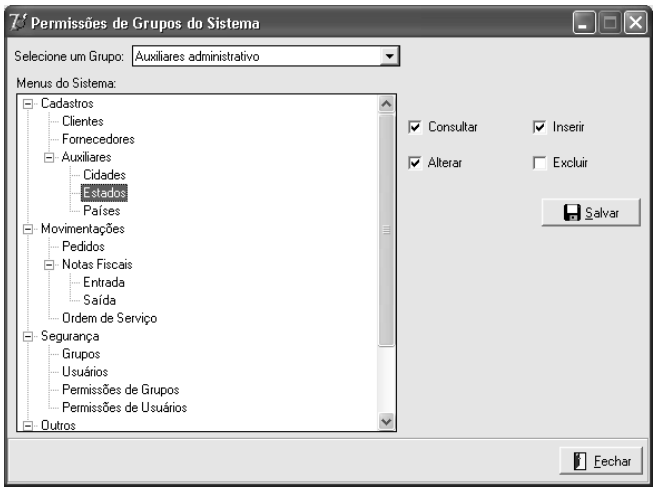


Figura 3. Exemplo de tela para definição das permissões de acesso aos grupos do sistema

Repare que no banco de dados definimos as chaves estrangeiras das tabelas de permissões com a cláusula DELETE CASCADE. Isso é necessário, pois caso algum item de menu for apagado do *mmPrincipal*, na próxima vez que o sistema for executado os registros relacionados a esse item nas tabelas de permissões serão apagados automaticamente.

Para podermos carregar os direitos de acesso do usuário logado no sistema referente aos itens de menu e telas, devemos declarar as seguintes variáveis na seção *public* do *FrmPrincipal*:

```
public
  codUsuario: Integer;
  codMenu: Integer;
  admin: Boolean;
```

Essas variáveis serão definidas e utilizadas em vários pontos do sistema, para que possamos controlar as permissões de acesso do usuário ativo. Vamos implementar agora a rotina que será responsável em chamar a *Stored Procedure* que fará a verificação dos direitos de acesso do

ClubeDelphi 76.indb 42 27.07.06 15:39:19

usuário logado nos itens de menu disponíveis no sistema. No *FrmPrincipal* digite o código da **Listagem 8** para o evento *OnShow* do formulário.

Listagem 8. Código do OnShow do formulário

```
procedure TFrmPrincipal.FormShow(Sender: TObject);
var
  i: Integer;
begin
  if admin then
    Exit;
  for i := 0 to Pred(mmPrincipal.Items.Count) do
    verificaAcessoSubMenu(mmPrincipal.Items[i]);
end;
```

No código da listagem anterior verificamos se a variável *admin* está definida como *True*, caso positivo, saímos do evento sem fazer a verificação de acesso aos menus do sistema, disponibilizando todos ao usuário, visto que o mesmo está definido no cadastro de usuários como Administrador.

Caso o usuário não seja Administrador realizamos um *loop* (*for..do*) nos itens do *mmPrincipal* passando cada item como parâmetro de entrada ao método *verificaAcessoSubMenu*. Na **Listagem 9** temos a implementação do método.

Listagem 9. Implementação do método verificaAcessoSubMenu

```
procedure TFrmPrincipal.verificaAcessoSubMenu(
  const menu: TMenuItem);
var
  i: Integer;
begin
  if not VerificarAcesso(retornarCodMenu(
    RetirarCaracterAtalho(menu.Caption))) then
    menu.Visible := False;
  for i := 0 to Pred(menu.Count) do
    verificaAcessoSubMenu(menu.Items[i]);
end;
```

O *verificaAcessoSubMenu* por sua vez faz a chamada a um método auxiliar denominado *VerificarAcesso* e na seqüência realiza um *loop* em todos os sub-itens do menu corrente fazendo uma chamada recursiva a ele mesmo.

O método *verificarAcesso* recebe como parâmetro de entrada o código do menu corrente. Utilizamos os métodos auxiliares *RetirarCaracterAtalho* e *retornarCodMenu* para capturar o código do menu em questão. Veja na **Listagem 10** a implementação do *verificarAcesso*.

Listagem 10. Implementação do método verificarAcesso

```
function TFrmPrincipal.VerificarAcesso(
  const codMenu: Integer) : Boolean;
begin
  with DMPrincipal.spAcessoMenu do
  begin
    Close;
    ParamByName('CODUSUARIO').AsInteger := codUsuario;
    ParamByName('CODMENU').AsInteger := codMenu;
    ExecProc;
    if (ParamByName('CONSULTAR').AsString = 'S') or
      (codMenu = -1) then
      Result := True
    else
      Result := False;
  end;
end;
```

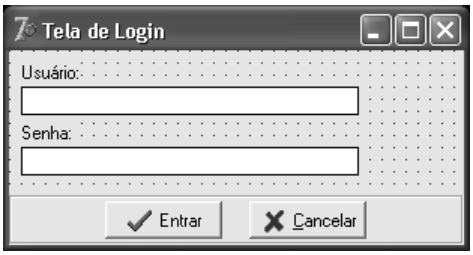


Figura 4. Tela de Login da aplicação de exemplo

O método é bastante simples, ele apenas faz a chamada a PROC_VERIFICA_ACESSO_MENU através do *spAcessoMenu* do *DMPrincipal* passando o código do usuário logado e o código do menu a ser verificado. Caso esse método retorne *False* alteramos a propriedade *Visible* do item de menu para *False* desabilitando assim o acesso ao mesmo pelo usuário.

Para que possamos testar os recursos de controle de acesso de nossa aplicação vamos criar uma tela de login. Crie um novo formulário e altere nome do mesmo para "FrmLogin". Salve sua *unit* como "untFrmLogin.pas". Adicione alguns componentes visuais e configure-os como mostra a **Figura 4.**

Declare as seguintes varáveis na seção *public* do formulário:

```
public
  codigoUsuario: Integer;
  nomeUsuario: string;
  administrador: Boolean;
```

Para o evento *OnClick* do botão *Entrar* digite o código da **Listagem 11.**

Listagem 11. Código para validar o usuário

```
procedure TFrmLogin.btnOKClick(Sender: TObject);
begin
  { Atenção no nome dos componentes Edits }
  if edtUsuario.Text = '' then
  begin
    MessageDlg('Informe o Usuário', mtInformation,
      [mbOk], 0);
    edtUsuario.SetFocus;
    Exit;
  end;
  if edtSenha.Text = '' then
  begin
    MessageDlg('Informe a Senha', mtInformation,
      [mbOk], 0);
    edtSenha.SetFocus;
    Exit;
  end;
  ValidarUsuario(AnsiUpperCase(edtUsuario.Text),
    edtSenha.Text);
end;
```

O código da listagem anterior faz a verificação do preenchimento dos campos da tela de login e realiza a chamada ao método *ValidarUsuario*, passando como parâmetro o usuário e senha informada. Veja na **Listagem 12** a implementação do método *ValidarUsuario*.

Listagem 12. Método ValidarUsuario

```
procedure TFrmLogin.ValidarUsuario(
  const usuario, senha: string);
begin
  { Declare em uses untDMPrincipal }
  with DMPrincipal.datasetAux do
  begin
    Close;
    CommandText :=
      'SELECT USU_CODIGO, USU_NOMECOMPLETO, USU_ADMIN'+
      ' FROM USUARIOS ' +
      ' WHERE UPPER(USU_LOGIN)= '+ QuotedStr(usuario)+
      '   AND USU_SENHA = ' + QuotedStr(senha);
    Open;
    if not IsEmpty then
    begin
      codigoUsuario := Fields[0].AsInteger;
      nomeUsuario := Fields[1].AsString;
      administrador := Fields[2].AsBoolean;
      ModalResult := mrOk;
    end
    else
    begin
      MessageDlg('Usuário ou senha inválida!',
        mtInformation, [mbOk], 0);
      edtUsuario.SetFocus;
    end;
  end;
end;
```

Mais uma vez utilizamos o *datasetAux* do *DMPrincipal* para enviar uma instrução SQL pra fazer a validação do usuário e senha. Para chamar a tela de Login, altere o arquivo de projeto *ControleAcesso.dpr* (menu *Project>View Source*) como mostrado na **Listagem 13**.

Listagem 13. Código do projeto para abertura da tela de Login

```
program ControleAcesso;

uses
  Forms,
  Controls,
  SysUtils,
  untFrmPrincipal in 'untFrmPrincipal.pas'
   {FrmPrincipal},
  untDMPrincipal in 'untDMPrincipal.pas'
    {DMPrincipal: TDataModule},
  untFrmLogin in 'untFrmLogin.pas' {FrmLogin};

{$R *.res}

begin
  Application.Initialize;
  Application.CreateForm(TDMPrincipal, DMPrincipal);
  FrmLogin := TFrmLogin.Create(nil);
  if FrmLogin.ShowModal = mrOk then
  begin
    Application.CreateForm(
      TFrmPrincipal, FrmPrincipal);
    FrmPrincipal.codUsuario := FrmLogin.codigoUsuario;
    FrmPrincipal.admin := FrmLogin.administrador;
    FreeAndNil(FrmLogin);
    Application.Run;
  end
  else
    Application.Terminate;
end.
```

> **Nota:** Para mais informações sobre a criação e utilização de telas de login acesse o artigo *Utilizando tela de login* disponível no Portal do Assinante (*www.clubedelphi.net/portal*).

Para testar a aplicação adicione um usuário no banco de dados, defina alguns direitos de acesso ao mesmo e rode a aplicação. Veja a aplicação de exemplo rodando na **Figura 5.**

Para finalizar o exemplo vamos habilitar/desabilitar os

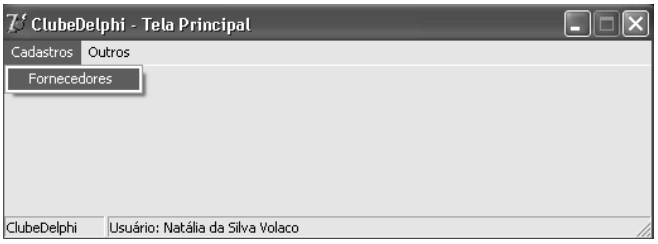


Figura 5. Aplicação de exemplo em execução mostrando apenas os menus disponíveis para o usuário logado

Figura 6. Formulário para cadastro de clientes

ClubeDelphi 76.indb 44 27.07.06 15:39:20

botões de uma tela de cadastro, obedecendo os direitos de acesso definidos no banco de dados. Crie um novo formulário e altere o nome do mesmo para "FrmCadClientes". Salve sua *unit* como "untFrmCadClientes.pas",adicione alguns botões ao formulário e configure-os como mostra a **Figura 6.**

Implementaremos nessa tela apenas as funcionalidades referentes ao controle de acesso aos botões. Declare as seguintes variáveis na seção *private* do formulário:

```
private
  novo, alterar, apagar: string;
```

Digite o código da **Listagem 14** no evento *OnCreate* do *FrmCadClientes*.

Listagem 14. Verificando os direitos de acesso do usuário

```
procedure TFrmCadClientes.FormCreate(Sender: TObject);
begin
  { Declare em uses untDMPrincipal e untFrmPrincipal }

  with DMPrincipal.SPDireitoAcesso do
  begin
    Close;
    ParamByName('CODUSUARIO').AsInteger :=
      FrmPrincipal.codUsuario;
    ParamByName('CODMENU').AsInteger :=
      FrmPrincipal.codMenu;
    ExecProc;
    novo := ParamByName('NOVO').AsString;
    alterar := ParamByName('ALTERAR').AsString;
    apagar := ParamByName('APAGAR').AsString;
  end;

  { Atenção ao nome dos componentes }
  btnNovo.Enabled := (novo = 'S');
  btnAlterar.Enabled := (alterar = 'S');
  btnApagar.Enabled := (apagar = 'S');

end;
```

No código da listagem anterior fazemos a chamada à SP PROC_VERIFICA_DIREITOS_ACESSO através do *spDireitoAcesso* do *DMPrincipal*. A partir do resultado retornado na execução da SP habilitamos ou não os botões *Novo*, *Alterar* e *Apagar*.

Conceda o direito de acesso *Consultar* a tela de Cadastro de Clientes ao seu usuário de teste e rode novamente a aplicação. Entre na tela de Clientes e verifique o estado dos botões. Para fazer a chamada aos formulários da aplicação a partir dos itens de menu, utilizei a função *CriarForm* que centraliza a chamada aos formulários.

Antes da chamada dos formulários é necessário preencher a variável *codMenu*, com o seguinte código:

```
codMenu := retornarCodMenu(RetirarCaracterAtalho(
  (Sender as TMenuItem).Caption));
```

> **Nota:** Para obter mais informações sobre a centralização das chamadas aos formulários da aplicação, acesse o artigo *Chamando Formulários* disponível no Portal do Assinante.

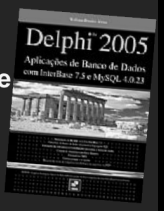
Conclusão

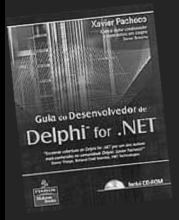
Vimos neste artigo uma maneira interessante e flexível de controlar os direitos de acesso às opções de nossa aplicação. Você pode facilmente implantar e até mesmo aperfeiçoar essa idéia de controle de acesso em suas aplicaçõesDelphi. Um grande abraço e até a próxima. ■

LIVROS DE DELPHI

Estudo Dirigido de Delphi 2005
Editora Érica
R$ 76,00

Delphi 2005: Aplicações com Banco de Dados com Interbase 7.5 e MySQL 4.0.23
Editora Érica
R$ 145,00





Guia do Desenvolvedor de Delphi for .NET
Makron Books
R$ 165,00

Rave Report com Delphi
Ciência Moderna
R$ 25,00

Universidade Delphi
Digerati Books
R$ 49,90

Delphi: Progra. Banco de Dados e Web - R$69,00
Delphi 7 & E-commerce: Trein. Inter. CD - R$381,00
Delphi - R$39,00
Aplic. das Estruturas de Dados em Delphi - R$59,00
Impl. .NET no Delphi 8: Uma Visão Geral - R$31,00
CD com 50 Program./Ex. Fon. p/ Delphi - R$29,90
Delphi 8 Para Plat. .NET: Curso Completo- R$249,00
Program. em Delphi 6: Orient. por Projeto - R$56,00
Int. ao Desen. de Aplicações em Delphi - R$50,00
Conhecendo e Trabalhando com Delphi 8 - R$80,00
Estudo Dirigido de Delphi 8 - R$58,00
Pr Pascal: Ling. do Turbo Pascal do Delphi -R$52,00
Kylix: Delphi Linux com Interbase/Firebird - R$52,00
Delphi 6: Conceitos Básicos (E-Book) - R$27,90
Acessando MySQL com Delphi 7 (em CD) - R$20,00
Dominando o Delphi 7: A Bíblia - R$189,00
Redes Neurais em Delphi - R$24,00
Estudo Dirigido de Delphi 7: Avançado - R$65,00
CLX Portabilidade com Delphi 7 e Kylix 3 - R$39,00
Delphi 7: Apl. Avan. de Banco de Dados - R$87,00
Delphi 7: Conceitos Básicos - R$42,00
Estudo Dirigido de Delphi 7 - R$74,00
Programação Gráfica em Delphi 6 - R$50,00
Delphi/Kylix: Desenv. de Banco de Dados - R$72,00
Boleto Bancário em Delphi - R$35,00
Conectividade Utilizando Delphi 6 - R$40,00

FRETE GRÁTIS
Para todo o Brasil!

LivrosdeProgramacao.com.br

VICTORY FERNANDES E MURILO PLINIO

Biometria

Reconhecimento de impressão digital com Delphi

VICTORY FERNANDES
(victory@igara.com.br) é Engenheiro Mestrando em Redes de Computadores, e desenvolvedor sócio da TKS Software - Soluções de Automação e Softwares Dedicados. Pode ser contatado em através dos sites www.igara.com.br - www.victory.hpg.com.br.

MURILO PLÍNIO
(muriloplinio@yahoo.com.br)
é estudante de Engenharia Mecatrônica, Técnico em Informática e desenvolvedor da TKS Software - Soluções de Automação e Softwares Dedicados. Pode ser contatado em

Seguindo a linha de publicações voltadas à integração de sistemas computacionais a dispositivos externos diversos, apresentamos neste artigo uma abordagem prática para integração de aplicativos a dispositivos biométricos, nesse caso específico, dispositivos de leitura de impressões digitais.

Um dos maiores problemas enfrentados atualmente em termos de segurança em informática é a autenticação de usuários, que implica em garantir que a pessoa que está tentando acessar um sistema é quem ela realmente diz ser.

Na maioria dos sistemas de informação, a autenticidade dos usuários é garantida através de senhas alfanuméricas que devem ser memorizadas pelos usuários. Apesar de ser a forma de autenticação mais difundida na atualidade, a autenticação por senhas apresenta diversas vulnerabilidades bastante evidentes.

Primeiramente senhas devem ser memorizadas. Quando não memorizadas, são comumente anotadas em local nem sempre seguro, ficando assim propensas a cair em mãos erradas. Em segundo lugar, senhas estão também sujeitas a observadores não autorizados durante a digitação.

Devido a esses problemas, técnicas de autenticação baseadas em características biométricas físicas, como impressões digitais, exames de retina e da palma das mãos vêm sendo cada vez mais utilizadas para garantir a autenticidade dos usuários.

Neste artigo serão abordados os conceitos básicos envolvendo a leitura, cadastro e autenticação de impressões digitais utilizando dispositivo de *hardware* FDU01 (Interface USB e resolução de 500DPI para uso em aplicações PC) da empresa Nitgen e banco de dados InterBase para o desenvolvimento de aplicações completas, com base na tecnologia de autenticação biométrica.

A escolha específica do dispositivo visa a exemplificação dos conceitos teóricos apresentados através de uma implementação prática e detalhada dos mesmos, não constituin-

ClubeDelphi 76.indb 46 27.07.06 15:39:22

do portanto restrição ao uso e aplicação dos mesmos em hardwares de outros fabricantes.

O artigo apresenta toda a estrutura de software e hardware utilizada e têm por objetivo familiarizar o usuário com os conceitos básicos que envolvem esse tipo de integração, fornecendo ao leitor uma visão ampla sobre o tema, suas características, desafios e principais problemas.

Para a leitura e entendimento deste artigo não se faz necessário um vasto conhecimento a respeito do processo de integração de dispositivos biométricos, uma vez que a API para implementação de tal integração é vastamente documentada pelo fabricante e acompanha diversos aplicativos de demonstração.

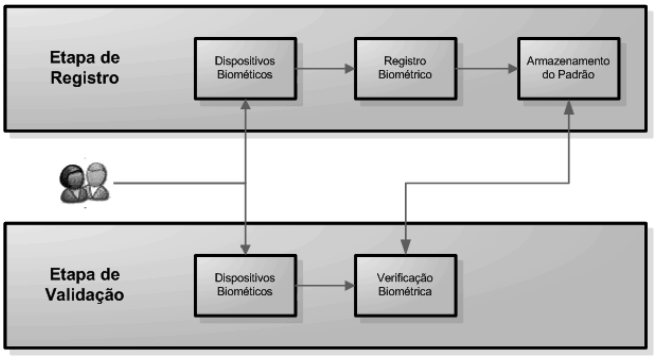


Figura 1. Etapas de Registro e Verificação dos processos biométricos

Biometria - Conceitos Gerais

Biometria é o conjunto de métodos automatizados para reconhecer uma pessoa com base em características comportamentais ou fisiológicas. São exemplos de características comportamentais: Escrita manual, assinatura, gestos.

São exemplos de características físicas: Face, impressões digitais, geometria da mão, íris. Este artigo é focado no que diz respeito à autenticação biométrica utilizando a característica física da impressão digital.

A autenticação de um usuário pode ser feita através de informações que o usuário conhece como senhas ou através de dispositivos que o usuário possua como cartões magnéticos, códigos de barras ou chaves. O que acontece é que em todos os casos citados, os dispositivos de autenticação podem ser perdidos ou roubados.

A biometria vem para tentar solucionar esse problema, pois, ao contrário das outras técnicas, ela se baseia no que você é, ou seja, em informações contidas em seu corpo ou com base no seu comportamento, dificultando e muito o processo de fraude.

Tipicamente o processo de uso de tecnologias biométricas passa por duas etapas bem definidas, o Registro e a Validação, conforme mostrado na **Figura 1**.

Durante o processo de registro, ocorre:

- a captura dos parâmetros biométricos, utilizando-se para tanto o dispositivo de captura adequado;
- a extração das características biométricas para a geração do padrão biométrico dos dados lidos;
- o armazenamento do padrão biométrico gerado em uma estrutura de banco de dados para consulta posterior.

Durante o processo de validação, ocorre:

- a captura dos parâmetros biométricos utilizando-se para tanto o dispositivo de captura adequado;
- a extração das características biométricas para a geração do padrão biométrico dos dados lidos;
- a comparação do padrão lido com as informações armazenadas no banco de dados de padrões biométricos.

Biometria por impressões digitais

No caso da autenticação biométrica por impressões digitais, o dispositivo de captura capta e analisa padrões na pele da ponta dos dedos, gerando um padrão biométrico baseado em uma das principais abordagens utilizadas: casamento de minúcias (método policial), casamento de padrões retilíneos, padrões moiré ou uso de ultra-som.

Um ponto interessante a ser observado sobre impressões digitais é que ela não se modifica durante todo o curso da vida. Portanto, o desenho digital de uma criança, por exemplo, sofre apenas uma alteração de tamanho em função de seu crescimento, de forma semelhante a uma ampliação fotográfica.

Algumas das vantagens do uso da autenticação biométrica por impressões digitais, quando comparada com outras formas de autenticação biométrica, são: baixo custo do hardware, tamanho reduzido dos dispositivos de captura e fácil integração dos dispositivos com a aplicação.

A seguir na **Tabela 1** é apresentado um comparativo das principais tecnologias de biometria existentes.

Método	Padrão codificado	Erros de identificação	Segurança	Principais Aplicações
Íris	Padrão da Íris	1/1.200.000	Alta	Acesso a sistemas de alta segurança
Impressão digital	Minúcias	1/1.000	Média	Universal
Mão	Tamanho, comprimento dos dedos e da mão	1/700	Baixa	Acesso a sistemas de baixa segurança
Face	Perfil, distribuição dos olhos, nariz, boca e características globais	1/100	Baixa	Acesso a sistemas de baixa segurança
Assinatura	Formato da letra, ordem da escrita, pressão da caneta	1/100	Baixa	Acesso a sistemas de baixa segurança
Voz	Características da voz	1/30	Baixa	Serviços de telefonia

Tabela 1. Comparativo das principais tecnologias de biometria existentes

ClubeDelphi 76.indb 47 27.07.06 15:39:24

A autenticação por impressão digital é hoje um dos meios mais difundidos de autenticação biométrica, principalmente devido ao relativo baixo custo de seus dispositivos, que hoje já se encontram embutidos em mouses, teclados e notebooks, permitindo a implementação de uma série de aplicativos baseados em suas funcionalidades.

A API Nitgen

O conjunto de funções oferecidas pelo fabricante é disponibilizado em dois níveis: API Primitiva e API em Alto Nível. Para o estudo e uso das APIs disponibilizadas faz-se necessário o conhecimento de algumas terminologias, dentre as quais destacam-se:

- FIR (*Fingerprint Identification Record*): registro de identificação de impressão digital. Um FIR é composto pelos campos de formato, cabeçalho e dados da impressão digital e armazena todas as informações necessárias à identificação biométrica;
- *Payload*: Espécie de chave associada a um FIR utilizada para transferência criptografada da identificação. É uma informação do tipo senha ou nome do usuário.

Caso necessário, o conhecimento das demais terminologias pode ser obtido através do manual oferecido pelo fabricante. Algumas aplicações são atendidas pelas funções em alto nível, tais como aplicativos *standalone* simples que não possuem conectividade Web ou validação através de estrutura cliente/servidor.

As funções da API em alto nível são implementadas utilizando as funções da API Primitiva e têm por propósito ser mais amigáveis e fáceis de usar, conforme vemos na **Tabela 2**.

Funções da API em Alto Nível	
Função	**Descrição**
Enroll	Usada para extrair pontos característicos de uma leitura e armazená-los. Essa função pode também receber um FIR antigo e gerar um novo registro (FIR) com um Payload associado.
Verify	Usada para confrontar uma impressão FIR lida com informações previamente armazenadas. O resultado é retornado pela função e, se existir um Payload associado, esse também é retornado.

Tabela 2. Descrição dos métodos da API em alto nível

Funções da API Primitiva

Já aplicações que, por exemplo, capturam dados da leitura num cliente e as confrontam com informações dos usuários num servidor, exigem funções descritas na API Primitiva. A API Primitiva permite o acesso direto a todos os recursos do equipamento, sendo necessário também um maior conhecimento para utilizá-la, conforme vemos na **Tabela 3.**

Funções da API Primitiva	
Função	**Descrição**
Capture	Utilizada para capturar uma imagem do dispositivo e extrair pontos que formarão um modelo. Diversas amostras são utilizadas para gerar tal modelo. Uma vez que o processo de captura é finalizado, a função retorna um registro de identificação (FIR) como resultado. A aplicação é quem determina o propósito da função que pode ser para registro, verificação ou identificação. Esse propósito é passado como parâmetro e gravado no cabeçalho do registro FIR.
VerifyMatch	Confronta um novo registro de identificação FIR com os dados previamente armazenados. O resultado da comparação é retornado pela função.
CreateTemplate	Processa uma amostra para construir um registro de FIR. Pode receber como entrada tanto um FIR intermediário como um definitivo. Essa função pode também receber um FIR antigo e gerar um novo registro (FIR).

Tabela 3. Métodos da API Primitiva

Uma forma de acesso a essas funções é a *NBioBSP.dll* oferecida pelo fabricante e que implementa todas as funções descritas nas tabelas anteriores. Para desenvolvedores que não se sentem familiarizados com o uso de DLLs, é interessante ressaltar que o fabricante também disponibiliza bibliotecas de classes para desenvolvimento em ambiente .NET e módulo COM.

Entretanto, neste artigo vamos nos ater à descrição do uso do dispositivo através da *NBioBSP.dll*. Maiores informações sobre COM e .NET podem ser encontradas na documentação disponibilizada com o produto.

Exemplo de integração com dispositivo biométrico

Agora que o leitor está familiarizado com a API dos dispositivos biométricos da Nitgen, apresentaremos o aplicativo desenvolvido para demonstrar o uso dessa tecnologia em conjunto com aplicações Delphi.

A aplicação proposta, que está disponível no endereço para download, visa demonstrar de forma prática os processos básicos de Registro e Validação descritos no tópico *Biometria - Conceitos Gerais* e demonstrado na **Figura 1**, através de um sistema de cadastro e autenticação de funcionários.

ClubeDelphi 76.indb 48 27.07.06 15:39:24

Inicialmente criamos a tabela no banco de dados que receberá as informações dos funcionários, tais como nome, foto, bem como informações recebidas do dispositivo que incluem a identificação do dedo e da impressão digital do usuário codificada em formato texto. A tabela é semelhante a **Listagem 1**.

Listagem 1. Tabela para armazenamento de informações do Funcionario

```
CREATE TABLE FUNCIONARIO
(
  CODIGO     INTEGER NOT NULL,
  NOME       VARCHAR(50),
  FOTO       VARCHAR(255),
  USERID     VARCHAR(10),
  FINGERID   VARCHAR(10),
  SAMPLENUMBERVARCHAR(10),
  TEXTENCODEFIRBLOB SUB_TYPE 0 SEGMENT SIZE 80,
 PRIMARY KEY ("CODIGO")
);
```

Uma vez criado o banco de dados, partimos para a criação do sistema. Na **Figura 3** temos a tela inicial do sistema, onde se pode cadastrar as informações de nome e foto do funcionário no banco de dados através de um *DBGrid*.

Após salvar as informações cadastradas de nome e foto do funcionário, devemos cadastrar as informações relativas aos dados biométricos do mesmo. Para isso, basta selecionar o funcionário desejado no *DBGrid* e clicar o botão *Cadastra Digital*. Nesse momento surge a tela mostrada na **Figura 4** (obviamente se você tiver o equipamento instalado).

Essa tela faz parte da API Nitgen e permite que o usuário selecione o dedo a ser utilizado na captura dos dados biométricos do funcionário. Após selecionar o dedo desejado clicando em um dos dez círculos azuis disponíveis, surge a tela mostrada na **Figura 5.**

Ao entrar nessa tela, o sistema passa a aguardar a leitura de uma impressão digital e os *leds* do dispositivo piscam até que o funcionário coloque o dedo sobre o leitor. Ao colocar o dedo sobre o leitor o sistema faz uma primeira amostra dos dados biométricos do funcionário e passa a aguardar uma segunda leitura para comparar com a primeira leitura e confirmar a validade das informações lidas.

Nesse momento podem acontecer duas situações distintas:

1. A segunda amostra pode não guardar semelhanças suficientes com a primeira, seja por má leitura, ou seja, porque os dedos lidos na primeira e na segunda amostragem não coincidem. Nesse caso é informando erro na comparação das leituras;
2. A segunda amostra coincide com a primeira. Nesse caso o sistema apresenta por alguns instantes a tela da **Figura 6** indicando que está gerando a codificação da impressão digital, e logo em seguida será realizado o fim do processo de registro de informações biométricas.

Após o término do processo, o aplicativo é responsável por editar o registro da base de dados armazenando as informações codificadas do funcionário nos seus respectivos campos. A codificação em formato texto da digital é

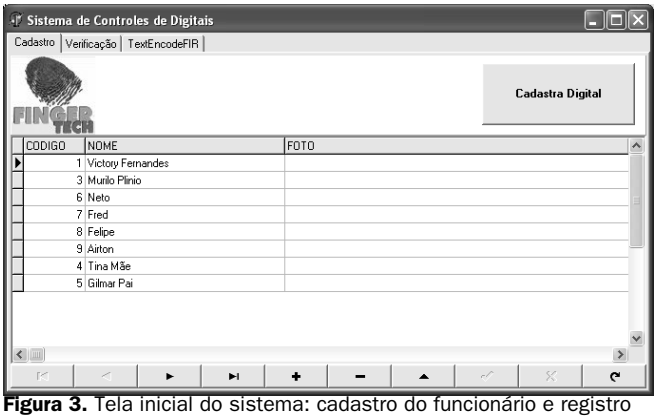


Figura 3. Tela inicial do sistema: cadastro do funcionário e registro da impressão digital

Figura 4. Seleção do dedo cuja digital será registrada

Figura 5. Primeira amostragem da digital a ser cadastrada

ClubeDelphi 76.indb 49 27.07.06 15:39:25

armazenada no campo TEXTENCODEFIR, do tipo BLOB, pois as informações armazenadas nesse campo serão semelhantes ao mostrado na **Listagem 2**, obtidos a partir da leitura de uma digital:

Listagem 2. Valor salvo no campo

```
AQAAABQAAAAkAQAAAQASAAMAZAAAAAAAAIAEAAOl32Zcf5RvZOH6OK9
sI5dIXgZGNqOeM6TOD/CG2g6P/YF4YtIsOuYtcftn7FjUSA7pRgJIR
xNeoIFxBR6w5rUhwqlzSh4E5*YY*NhxPB3WcU4b7WevT33uuw0q6Ak
vNF17uMw7GriI8/HsIhmAVxW4N4umjWx93f3UgA4/sDhZMQPM01wQe
M8Eh8BE4DT*FUp1LzObHxOb7eKK37qZJIr8e*0kNBg1/ENmFmQOiel
ZDl0Cxbe*c*N36ZxLs5x*34YpKndHycG1mU9hOhzKj5xdL2VPWefI3
lijDXp6RDyXL*1ro5nrbNU2PLjVF8QIYIVjhlK4K/aM24JqX0q0qSp
GqEpIqupakzgW4VCk34BI2yJQlf0MoUME1uoPSxiQUmg
```

Depois de concluída a etapa de cadastro e registro das informações biométricas do funcionário, avançamos para a etapa de validação, mostrada na **Figura 7.**

Ao clicar no botão *Verifica Digital*, surge a tela para que o usuário coloque o dedo no equipamento para leitura biométrica. Essa tela faz parte da API Nitgen e permite a leitura dos dados biométricos do funcionário. Ao entrar nessa tela, o sistema passa a aguardar a leitura de uma impressão digital e os *leds* do dispositivo piscam até que o funcionário coloque o dedo sobre o leitor.

Ao colocar o dedo sobre o leitor, o sistema faz a leitura dos dados biométricos do funcionário, codifica os dados lidos e compara com as informações cadastradas previamente no banco de dados.

Nesse momento, o sistema deve ser capaz de distinguir entre as impressões digitais de funcionários cadastrados e não cadastrados, apresentando uma mensagem "funcionário não cadastrado", caso a impressão digital lida não coincida com nenhuma das impressões digitais contidas no banco de dados. Caso a impressão digital lida coincida com a impressão digital de algum funcionário, a foto do respectivo funcionário é apresentada.

Conclusão

Com este artigo abordamos conceitos gerais de biometria bem como toda a teoria e prática necessária ao desenvolvimento de aplicações integradas a dispositivos de autenticação biométrica por impressões digitais da *Nitgen.*

No entanto, apesar das informações apresentadas envolverem dados específicos do fabricante, os conceitos abordados são semelhantes no que diz respeito a comunicação com dispositivos de autenticação biométrica por impressões digitais, sendo assim o conhecimento adquirido pode ser de grande utilidade na implementação da comunicação com outras marcas e modelos.

O aplicativo de demonstração apresentado não cobre todos os comandos disponíveis na API do dispositivo, mas abrange todos os principais conceitos necessários para que você se familiarize com a tecnologia e inicie o seu trabalho a partir desse ponto. Afinal, você nunca poderia imaginar que seria tão fácil agregar valor aos seus aplicativos através da integração de soluções de biometria.

Figura 6. Amostragens aceitas, codificação da impressão digital

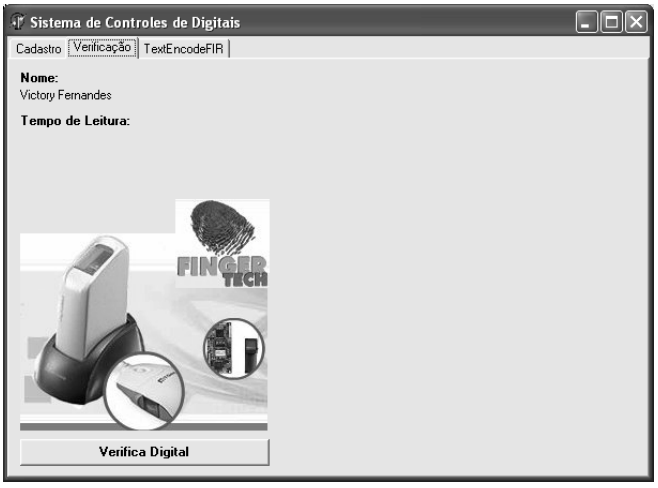


Figura 7. Etapa de validação: autenticação das impressões digitais

Referências

Riha, Zdenek; Matyas, Vaclav. *Biometric authentication systems.* FI MU *Report Series*, RS-2000-08, November 2000.

Milton Roberto Heinen, Fernando Santos Osório. Biometria Comportamental: Pesquisa e desenvolvimento de um sistema de autenticação de usuários utilizando assinaturas manuscritas. *UNISINOS* - Universidade do Vale do Rio dos Sinos.

Saiba mais sobre a Biometria e como ela será usada na instituição. Faculdade *Seama*, 2005.

Prof. Adilson Gonzaga. Biometria, a senha que não se esquece. ■

Links

Nitgen Soluções biométricas

www.nitgen.com.br

ClubeDelphi 76.indb 50 27.07.06 15:39:26

Assinatura

msdn magazine PLUS

Mais conteúdo .NET por menos!

Sua assinatura da MSDN Magazine virou **MSDN Plus** e com isso você ganhou **470 vantagens!**

A MSDN PLUS é a combinação perfeita de conteúdo impresso, digital e áudio-visual. A partir de agora você terá acesso a um material completo de alta qualidade técnica, deixando você por dentro de tudo que acontece na plataforma .NET.

MSDN PLUS = Revista MSDN + Video Aulas + Artigos Online!

Você assinante já pode conferir este lançamento da MSDN Magazine. Com o seu login e senha você terá acesso imediato ao MSDN Plus. Hoje você já pode conferir mais de 90 vídeo-aulas e 20 artigos online. E ao longo de sua assinatura, todos os meses você terá acesso a mais 10 vídeo-aulas e 05 artigos online novos.

São cerca de 470 atualizações sobre .net durante um ano. Confira agora mais essa vantagem de ser assinante MSDN Magazine.

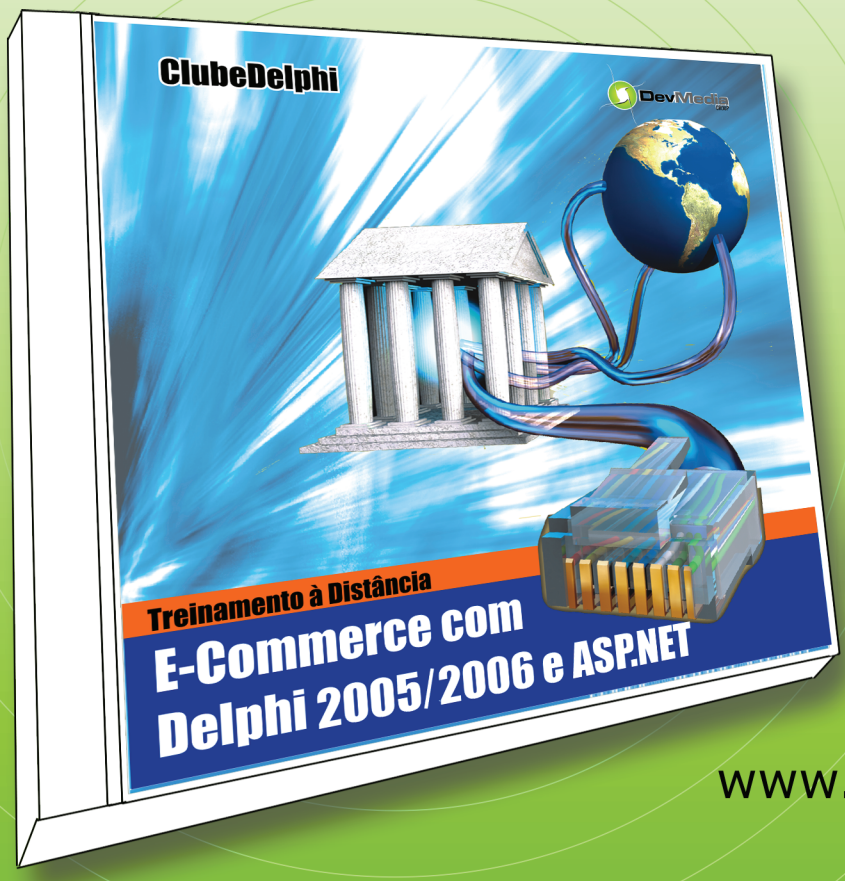
Por Apenas

6x de R$10,00

Saiba mais! Acesse: www.devmedia.com.br/msdn/msdn_plus.html

Treinamentos a distância.

Aprenda de forma prática, eficiente e econômica.



E-Commerce com Delphi 2005/2006 e ASP.NET

Aprenda a desenvolver poderosas aplicações Web através do Delphi 2005/2006 e as tecnologias ASP.NET, WebBroker, Intraweb e WebSnap.
O curso conta com módulos textuais;
35 horas de vídeo-aulas;
fórum exclusivo para os alunos e suporte on-line.

COMPRE JÁ O SEU!

www.devmedia.com.br/curso

Outros Cursos

Delphi e Relatórios

Com este curso você desenvolverá relatórios através do Delphi 7, do Delphi 2005 e Delphi 2006, utilizando as ferramentas Rave Reports, Quick Reports e Crystal Report. Além do módulo textual o curso contém mais de 20 horas de vídeo-aulas; fórum exclusivo e suporte

Evento à distância - Introdução ao .NET

Conheça os recursos do ASP.NET com o Visual Studio 2003 e aprenda a criar páginas ASP.NET; Webservices; trabalhar com a IDE do VS; componentização; DataGrid e criar aplicações de BD com o ADO.NET! São 6h de palestras em formato de vídeo-aulas e você pode tirar suas dúvidas com os palestrantes!

Evento à distância - Novidades do Delphi 2005

Conheça os recursos do Delphi 2005 e aprenda a criar páginas ASP.NET; Webservices; trabalhar com a nova IDE do Delphi; utilizar programação OO no Delphi 2005; trabalhar em equipe com o novo Starteam e criar aplicações de BD com o ADO.NET!

DevMedia